TC-30MR Study Group Report

Study of UMID Applications in MXF and Streaming Media

This SMPTE study group report is the work of TC-30MR.

The report discusses the Unique Material IDentifier (UMID) applications specifically for the material eXchange Format (MXF) technology.

TC-30MR Study Group Report

Study of UMID Applications in MXF and Streaming Media

Executive Summary

This SMPTE TC-30MR study report discusses the UMID applications specifically for the MXF technology.

The UMID, or Unique Material IDentifier, is the SMPTE standard unique identifier of an audiovisual material. Based on the recommendations by the EBU/SMPTE Task Force envisioning the file-based media production workflow we have today, it was initially SMPTE standardized in 2000 prior to the MXF technology in 2004.

As a mandatory component of an MXF file, the UMID has been widely spread in the Media & Entertainment (M&E) industry by using the MXF file as a vehicle. But it has been eventually useless in practice so far because of lack of industry standard technologies called UMID Application Principles and UMID Resolution Protocol.

The UMID Application Principles are fundamental rules for the UMID to be treated in a reliable and consistent way over the media products from multiple vendors, which have been standardized in the latest SMPTE RP 205 in 2014.

The UMID Resolution Protocol is a standard method for a given UMID to be converted into the corresponding URL of a material uniquely identified by the UMID, for which an intensive development is ongoing in SMPTE.

In this study report, based on those standard technologies, the UMID applications specifically for the MXF technology are discussed. Thanks to the MXF's logical structure, which realizes what we can observe at the playout of an MXF, file is not always identical with what is stored within the MXF file; an MXF file has three kinds of UMID. The first two are called MXF Material Package UID and File Package UID, both of which take a form of the 32-byte Basic UMID.

The MXF Material Package UID, or MpUmid, uniquely identifies an audiovisual material we will observe when an MXF file is played out. It is therefore suitable to be used as a globally unique material identifier for the MXF file, which will be used to associate the MXF file with its external metadata such as that describing the content of the file, and thus it needs to be appropriately managed based on the UMID Applications Principles.

The MXF File Package UID, or FpUmid, uniquely identifies an audiovisual material stored within an MXF file. While its primary use is for the File Package to be linked to the Material Package for its playout, it is also used as a linking tool that references back to the original material existing elsewhere, from which an internal material uniquely identified by the FpUmid within a MXF file derives.

The last one is called Body UMID, which takes a form of either the 32-byte Basic UMID or the 64-byte Extended UMID with so called Source Pack being appended. The Body UMID, or BodyUmid, is interleaved within a file body. More specifically, the BodyUmid is assigned to every frame stored in the MXF Essence Container to describe each frame, including when, where and who has initially created the frame by using the Source Pack.

Just like the UMID to be embedded into the SDI VANC at the VTR playout, the BodyUmid can go with a frame over the network at the playout of an MXF file. As a result, the information on the frame provided in the form of BodyUmid propagates over the media production workflow chain. In addition, because the basic part of BodyUmid to be assigned to a frame in a newly created MXF file is equivalent with the MpUmid globally uniquely identifying the MXF file as a whole, the original MXF file that contains the origin of any frame existing in the workflow chain can be unambiguously identified.

 Approved 11 December 2015

Furthermore, the UMID attached to a frame has a field to accommodate the number of media compression and decompression applied to the frame since its origin, which has been conventionally used for the copy number management in VTR. With a proactive employment of the field not only for the media compression/decompression but also for any other media processing applied to a frame even on the wire, the number of media processing applied to the frame since its origin can be managed for every frame in the media production workflow chain.

Such a seamless treatment of the BodyUmid assigned to a frame stored in an MXF file and the UMID attached to a frame on the wire can constitute a self-consistent and well-harmonized identification schemes for a media file and a media stream resulted from the playout of the media file, which must bring valuable opportunities for new applications in the modern file- and IP-based media production environment.

This study report is disclosed in public to seek feedback comments from the industry experts. Hence, those interested in this topic are kindly requested to review this study report carefully and to provide any feedback comments to the SMPTE Standard Community and/or the Study Group Chair indicated below. They all are to be respectfully taken into consideration when we explore the interoperable points that need to be confirmed by new SMPTE standard technologies in order to further enhance the UMID applications in MXF.

Yoshi Shibata (yoshi.shibata@metafrontier.jp)

Chair, SMPTE TC-30MR Study Group, UMID Applications

 Approved 11 December 2015

Table of Contents

 Approved 11 December 2015

 Approved 11 December 2015

1 Introduction

The UMID (Unique Material IDentifier) is a globally unique audiovisual material identifier standardized by SMPTE as SMPTE ST 330 [1] and RP 205 [2].

In 1998, the EBU/SMPTE Task Force for Harmonized Standards for the Exchange of Programme Material as Bitstreams recommended that a unique material identifier and a single, generic, file wrapper were needed to facilitate the exchange of program material and metadata as bit-streams [3]. These recommendations were realized with the standardization of the UMID in the year 2000 and later, with the standardization of the Material eXchange Format (MXF) in the year 2004 [4].

While the UMID itself is independent of any material format or packaging, it was a natural consequence from its origin that the UMID was adopted as the mandatory component in an MXF file by which the file, and the material it contains, is uniquely identified. Thus, as MXF has gained an ever-increasing role within the file-based media production workflows, the UMID has also become more widely spread within the industry by using the MXF file as a vehicle.

On the contrary to the original intent upon its introduction, however, the UMID has been unfortunately substantially useless in practice, *i.e.*, even the most expected role of the UMID as a globally unique material identifier to associate the material with its external metadata has seldom been seen in practice so far. According to the literature [5], this is because of lack of additional technologies needed to be industry standardized to enhance the UMID applications, which are called UMID Application Principles and UMID Resolution Protocol.

The UMID Application Principles are fundamental rules every UMID-aware product must strictly follow in order for the UMID to be treated in a reliable and consistent way over the media products from multiple vendors in the “best-of-breed” media production system, which have been recently SMPTE standardized as the latest SMPTE RP 205 in the year 2014 [2].

The UMID Resolution Protocol, on the other hand, is an industry common method for a given UMID to be converted into the corresponding URL of a material uniquely identified by the UMID. At the time of this writing, an intensive work is still on-going for it to be SMPTE standardized based on a relevant study report [6].

Both the UMID Application Principles and the UMID Resolution Protocol are agnostic to the format kind of a media file by their nature. Taking account of the origin of MXF and UMID described above, however, it is easily expected that an MXF file can take full advantage of the UMID applications based on those SMPTE standard technologies compared with media files of other file formats.

In this study report, we will explore how the UMID in an MXF file is to be proactively utilized to realize various UMID applications based on them including its application in a media streaming resulted from the playout of an MXF file. Having seen from another aspect, this can be regarded as a trial on how the UMID Application Principles are to be embodied specifically in the context of the MXF technology.

This study report is composed of five sections. After this introduction, an overview of the MXF technology is introduced in Section 2 with its internal behavior model at playout, followed by the UMID application basics as a reference of the UMID applications in MXF in Section 3. Then, the main body of this study report starts in Section 4 with the introduction of three kinds of UMID contained in an MXF file and specific applications of those respective UMIDs in various cases of the MXF file treatment. Finally, some plausible UMID applications in MXF, including those for the media streaming resulted from the playout of an MXF file, are schematically demonstrated and discussed as a consequence of this study report in Section 5.

 Approved 11 December 2015

2 MXF Overview

2.1 Introduction

An overview of the MXF technology is briefly introduced in this section. Although the MXF by itself comprises much elaborated file format specifications, only its basic characteristics and those needed in the following discussions are described. Readers requesting for a full knowledge of the MXF specifications are kindly guided to the MXF Engineering Guideline [7] as well as the MXF File Format Specifications [4] and its accompanied document suite.

2.2 What is the MXF?

2.2.1 Structure of an MXF file

The MXF, or Material eXchange Format, is a container format standardized as SMPTE ST 377-1 [4] and its accompanying document suite. The MXF is a container in the sense that an MXF file contains any kinds and any forms of audiovisual essences in it, from a single picture to audiovisual interleaved essence, from compressed to the uncompressed essence, and more. The MXF file also carries metadata such as timecode, titles and much more, which is to be associated with the whole of, or particular points of the audiovisual essence it contains.

An MXF file, in its physical representation, is composed of a sequence of SMPTE KLV (Key-Length-Value) coded chunks of data (called KLV packet hereafter) [7], which logically form a File Header starting with the Header Partition Pack, a File Footer starting with the Footer Partition Pack, and the File Body placed between them, as shown in **Figure 1** below.

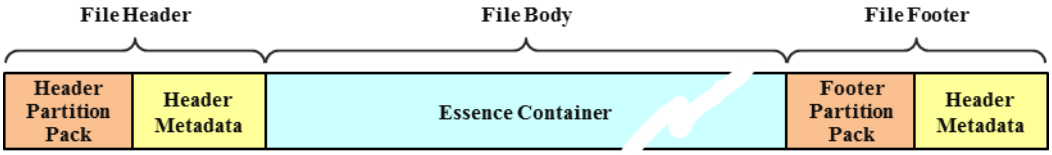


Figure 1: Simple MXF file [7]

In a typical MXF file, the essence data are individually KLV wrapped to form so called Essence Container, and stored in the File Body. The granularity of the essence to be KLV wrapped varies depending on the type of MXF file, *i.e.*, ranging from per frame (frame wrapped) to an entire essence (clip wrapped). In the case of the frame wrapped essence, metadata associated with a frame such as timecode is also attached to the KLV packet containing the frame.

2.2.2 Structural and Descriptive Metadata

Metadata associated with the whole file, or parts of the file, is basically contained in the File Header and additionally in the File Footer if desired, and is called the Header Metadata. Two kinds of Header Metadata are defined in MXF; the Structural Metadata and the Descriptive Metadata. The Structural Metadata describes one or more essence types and their relationship along a timeline at their playout. The Structural Metadata is mandatory because it describes how to play out the essence(s) by specifying their synchronization and technical parameters such as the frame rate.

Because the essence contained in the File Body is typically pure video and/or audio digital data without temporal information, there is no way for the essence to be played out without the Structural Metadata. Consequently, the modification of Structural Metadata can give a strong impact on the playout result of an MXF file.

The Descriptive Metadata provides information to be intended mainly for human use. Typical items in the Descriptive Metadata include a title, content synopsis, and so on. While it is also important for a user to search a desired MXF file efficiently, it is not mandatory in the sense that even an MXF file without the Descriptive Metadata can be played out without any problem. Hence, although it is crucial in the metadata based MXF applications in general, we don't go into more details for it in this study report.

2.2.3 Logical structure of MXF Header Metadata

The Header Metadata by itself is also a sequence of the KLV packets but they logically form a tree structure by connecting them via so called strong reference. **Figure 2** below shows a simplified logical structure of the MXF Header Metadata (the Structural Metadata).

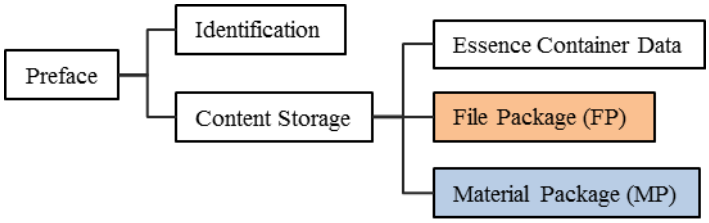


Figure 2: Logical structure of MXF Header Metadata

In this figure, the Preface as a root of the Header Metadata signals the kind of MXF file such as a type of MXF Operational Pattern (See Section 2.4) as well as the types of Essences Container including essence kinds contained in this MXF file.

Below the Preface locate the Identification and the Content Storage. The Identification describes which media product produces this MXF file and when, and the Content Storage contains three fundamental metadata items, the Essence Container Data, the Material Package and the File Package[1].

The Essence Container Data describes individual Essence Container in the File Body. Theoretically, an MXF file is permitted to contain more than one type of essence, each of which is stored in a specific Essence Container, which can be divided into its segments, and scattered within the File Body as needed. For example, an MXF file might contain the high resolution picture essence and its proxy altogether, where they are interleaved per frame basis within its File Body. In such a case, two instances of Essence Container Data are created; each of which bundles the relevant segments in the File Body to form an original Essence Container before the segmentation by using so called Body SID shared by the segments for each Essence Container.

According to SMPTE ST 377-1 [4], the Material and File Packages are specified to describe the essence on a timeline at the playout of an MXF file (called “the output timeline”) and the timelines for the essence(s) stored in an Essence Container (called “the input timeline”), respectively. In fact, this is one of the most characteristic parts of the MXF technology and also vital in the discussion of the UMID applications in MXF, which is therefore separately discussed in the next section.

2.3 MXF Internal Model

2.3.1 MXF with a dual layered structure

Among other valuable characteristics of the MXF technology, its dual layered structure would be one of the most distinguishing characteristics of the MXF. In short, what is to be produced at its playout is not always the same as what is contained in the File Body as shown in **Figure 3** below.

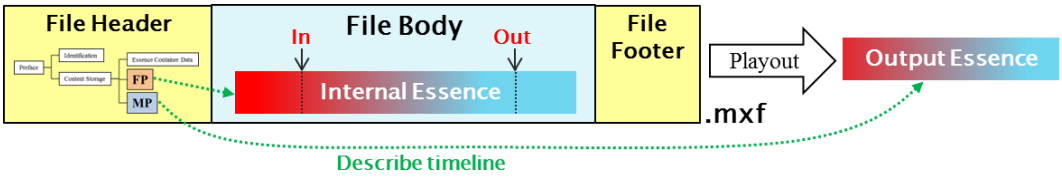


Figure 3: MXF at playout

In this figure, it is schematically demonstrated that only a middle part of the essence specified by the In/Out points in

[1] While this is also called the Top-Level File Packages or the Top-Level Source Packages in SMPTE ST 377-1 [4], we stick to use the term the File Package entirely within this study report.

the File Body (the Essence Container) is produced at its playout. The File Package (FP) and the Material Package (MP) describe the input and the output timelines, respectively. Or more specifically, the File Package attaches a timeline to a series of chunk of essence data in the Essence Container as needed while the Material Package specifies how they are to be placed on the timeline at playout. As a result, while the input timeline can contain discontinuity (*e.g.,* for the composite Essence Container), the output timeline must be always contiguous, though both of them are represented by using the timecode.

2.3.2 MXF internal behaviour model at playout

Another interesting point to be mentioned is that while the internal essence in the File Body can take various forms from the ones based on a variety of compression schemes to the uncompressed one, the output essence resulted from its playout is always the baseband signal that we can directly perceive (via devices such as a display monitor or the sound speakers, of course). Taking the roles of the File and the Material Packages into consideration, an MXF internal behavior that will occur at its playout is schematically demonstrated in **Figure 4** below.

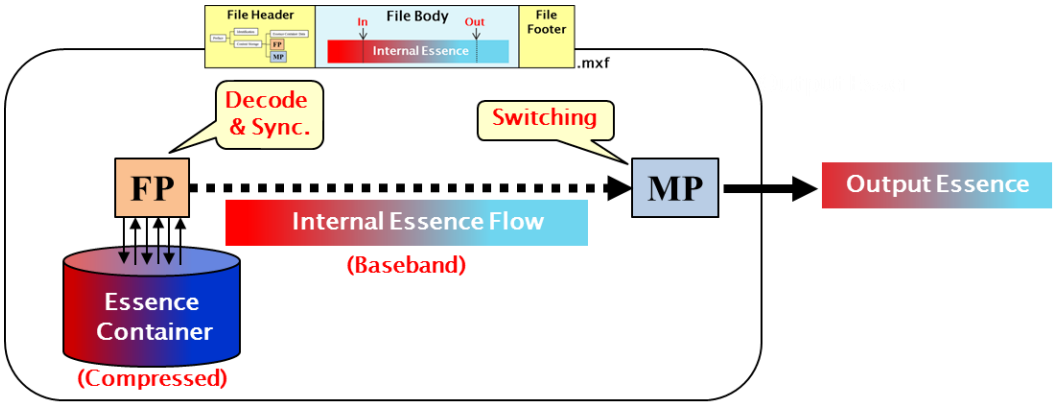


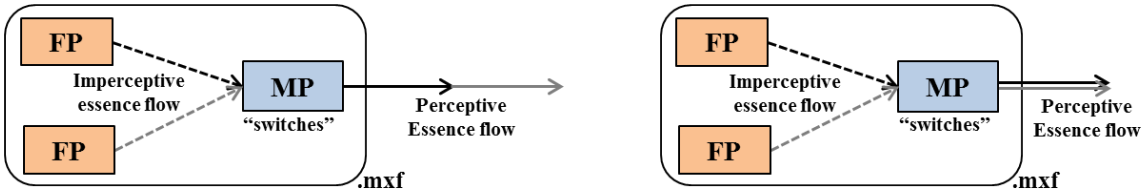
Figure 4: MXF internal behavior model at playout

Based on this figure, the MXF internal behavior model is described as follows: the essence data, encoded based on a certain compression scheme and stored in the Essence Container, is fetched and decoded by the File Package (FP). After the synchronization if more than one type of essence is decoded, the File Package supplies the baseband essence flow to the Material Package (MP), which is then switched by the Material Package so that only the part specified by the In/Out points is produced.

Because of such roles of the Material and the File Packages, the following features are observed:

- The File Package describes not only the temporal information of an Essence Container but also technical properties of the essence data stored in it such as codec, sampling rate (frame rate), frame size, and so on, by using so called File Descriptor, so that it can provide the internal essence flow as a baseband signal,
- The Material Package describes not only the temporal information of an essence at playout but also the switching information for it so that it can control the essence flow to be actually produced to output. Note that no File Descriptor is attached to the Material Package because it receives the (already decoded) baseband signal from the File Package as input,

It ought to be noted that the Material Package can references to one or more File Package as input and fully control the baseband essence flow(s) supplied by the File Package(s) for their actual output. For example, if a couple of baseband essence flows is supplied by two independent File Packages, the Material Package can either concatenate them for their sequential output or synchronize them for their parallel output as illustrated in **Figure 5** (a) and (b), respectively.



(a) For sequential output (b) For parallel output

Figure 5: Role of Material Package

Note that the baseband essence flow supplied by the File Package to the Material Package is *imperceptive* and thus cannot be accessed from an external application because it is just a conceptual essence flow derived based on the MXF internal behavior model discussed here in this section. On the other hand, the essence flow produced to output by the Material Package is perceptive, *i.e.*, it is what we actually observe at the playout of an MXF file. Hence, one can say that another important role of the Material Package is to make the imperceptive essence flow received from the File Package perceptive, based on the MXF internal behavior model.

2.4 MXF Operational Patterns

2.4.1 Introduction

The MXF technology has been designed to cover a wide range of applications, from a single clip to a multitude of clips and effects. But it is too demanding for an application requiring only the simplest MXF file to support all levels of complexity. To maximize the interoperability of MXF uses, so called Operational Pattern (OP) is introduced to define the constrained levels of an MXF file complexity. In this section, some types of OP are discussed from the viewpoint of the MXF internal behavior model introduced in Section 2.3.2.

2.4.2 Operational Pattern 1a (OP1a)

While there are nine basic types of Operational Patterns along with two axes, the Item and the Package Complexities, to form a matrix, the simplest one called OP1a [8] is the most widely available in the industry at the time of this writing.

The MXF OP1a file is the simplest MXF file where an internal essence described by the File Package is supplied to the Material Package as is, which is then reproduced to output exactly as is by the Material Package. **Figure 6** below schematically illustrates the MXF internal behavior for the MXF OP1a file.

Figure 6: MXF OP1a internal behavior model

In a typical MXF OP1a file, video and audio essence data for each frame are KLV wrapped and interleaved to form an audiovisual stream, which, as a whole, is stored in a single Essence Container. Therefore the File Package supplies the synchronized audiovisual essence as a single baseband essence flow to the Material Package, which is then reproduced to output exactly as is by the Material Package.

2.4.3 Operational Pattern 2a (OP2a)

As already mentioned in Section 2.3.2, more than one File Package can supply the baseband essence flow to the Material Package as shown in **Figure 5**, where **Figure 5** (a) and (b) correspond to the MXF OP2a [9] and MXF OP1b [10] files, respectively.

The MXF OP2a file contains two or more Essence Containers (and thus the File Packages describing each of them) and one Material Package that can only access a single Essence Container exactly as is at one time via the File Package describing it. Consequently, the Material Package produces the essence flow composed by sequentially concatenating the baseband essence flows supplied by those File Packages. **Figure 7** below schematically illustrates

 Approved 11 December 2015

the MXF internal behavior for an example of the MXF OP2a file containing two Essence Containers, each of which stores independent video essence data in it.

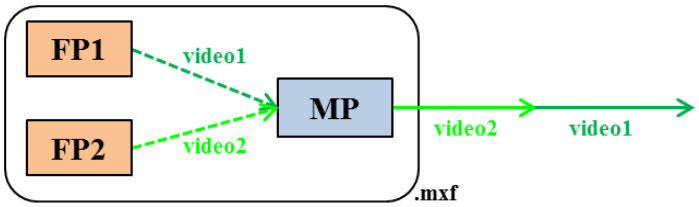


Figure 7: MXF OP2a internal behavior model

In this example, the baseband essence flow “video1”supplied by the File Package “FP1” is reproduced to output exactly as is by the Material Package, which is immediately followed by the baseband essence flow “video2” originally supplied by the File Package “FP2”. Note that while only the video essence is taken for example in **Figure 7** for simplicity, it could be audiovisual interleaved essence if it is stored in a single Essence Container as discussed in Section 2.4.2.

It ought to be noted that, in the MXF OP2a file, an entire essence stored in each Essence Container needs to be reproduced to output exactly as is, resulting in a simple concatenation of essences stored in multiple Essence Containers in the MXF file. In other words, if only a part of the essence specified by, *e.g.*, the In/Out points in each Essence Containers is desired to be concatenated, the MXF file needs to declare OP3a rather than OP2a, where the Material Package has a capability to specify a part of the Essence Container to be produced to output.

2.4.4 Operational Pattern 1b (OP1b)

The MXF OP1b file also contains two or more Essence Containers (and thus the File Packages describing each of them) and one Material Package. On the contrary to the MXF OP2a file, the Material Package in the MXF OP1b file can assess two or more Essence Containers at one time, resulting in describing the synchronization among essences from multiple Essence Containers. **Figure 8** below schematically illustrates the MXF internal behavior for an example of the MXF OP1b file containing two Essence Containers, one storing the video essence data and another storing the audio essence data.

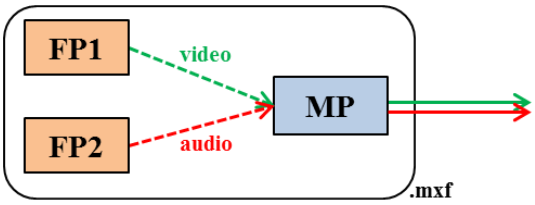


Figure 8: MXF OP1b internal behavior model

In this example, the baseband essence flows, “video” and “audio”, supplied by the File Package “FP1” and “FP2”, respectively, are synchronized and reproduced to output by the Material Package. Note that the essence flow concatenation discussed in Section 2.4.3 is prohibited for the OP1b, or it ought to be declared as OP2b if such a concatenation is desired.

It ought to be also noted that, while it looks as though the Essence Container storing either the video or audio essence is separately localized within an MXF file, e.g., it starts with the entire video Essence Container immediately followed by the audio Essence Container, this is not always necessary. Because an Essence Container can be divided into its segments and scatted within the File Body by using the Body Partition Pack as a separator, chunks of video and audio essence data that need to be synchronized can be placed closely within the file, resulting in their easy synchronization to be achieved at its playout.

2.4.5 AMWA AS-02 (AS-02)

While it is assumed in the example of **Figure 8** that a single MXF file contains the audio and video essence data altogether, it would be natural to consider the case where the audio and video essences are separately stored in different MXF files so that each of them can be independently treated.

Based on this consideration, AMWA (Advanced Media Workflow Association) has developed AMWA AS-02 (MXF Versioning) [11]. The AMWA AS series are specifications to constrain the use of MXF for specific purpose, and the AS-02 has been developed for easy repurposing of an MXF audiovisual program for multi-version, multilingual or multi-delivery of media environment.

In the AS-02, an audiovisual MXF program is decomposed into mono essences such as video essence, audio essence, and non-audiovisual data essence including subtitle, ancillary, etc., which are then stored into the respective MXF OP1a mono essence files (called Essence Component files) in a designated media folder. Their synchronization is then described by an MXF OP1b file (called Version file) located beside the folder containing the Essence Component files.

Figure 9 demonstrates how the MXF internal behavior for the MXF AS-02 files is to be schematically illustrated by taking the same example as shown in **Figure 8**. Compared with the **Figure 8** representation, the video and audio essences are stored in different Essence Component files and the Material Package that describes their synchronization is stored in the Version file.

According to SMPTE ST 377-1 [], while an MXF file is permitted to reference to external essence data instead of containing them within the file, it also requires at least one Material Package and one File Package to be always contained in any MXF file. Consequently, the AS-02 has adopted a strategy to let the Version file contain the File Package that describes the Essence Container existing externally. In other words, only the File Package instance, which is identical with the one the corresponding Essence Component file contains with its actual Essence Container, is stored into the Version file[2], resulting in a tiny size of the Version file compared with an Essence Component file.

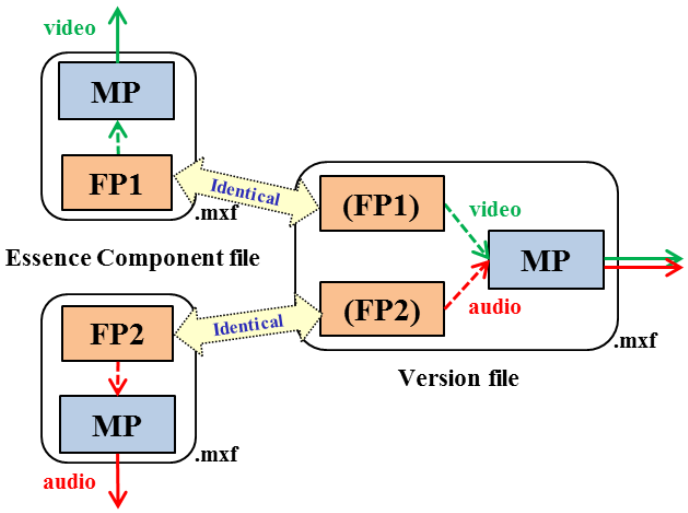


Figure 9: AMWA AS-02 internal behavior model

It ought to be noted that although the baseband essence flows supplied by the File Packages to the Material Package within the Version file is *imperceptive* as discussed in Section 2.3.2, each of them can be indirectly observed by playing out the Essence Component file of interest. This is because the Essence Component file is an MXF OP1a mono essence file that results in what is observed by its playout is identical with the baseband essence flow supplied by the File Package within the file, and therefore it is also identical with that in the Version file, according to the definition of the Version file.

[2] To unambiguously associate the File Package in the Version file with the one in the corresponding Essence Component file, the Network Locator for the Essence Component file is made required in the File Package instance of the Version file.

2.4.6 Operational Pattern "Atom" (OP Atom)

Whether audio and video essences ought to be stored by interleaving or separately depends on applications of interest. While an interleaved audiovisual essence stored in a single file such as an MXF OP1a file would be the most preferred one for the media file transfer or its playout, mono essence per media file is more convenient to use for the video edit desiring to manipulate individual essence independently.

Before the appearance of AMWA AS-02 MXF file, a specialized Operational Pattern called the MXF OP Atom [12] has been provided to meet the latter demand in addition to the MXF OP1a for the former one.

The MXF OP Atom files are a set of MXF files where each file is restricted to contain only a mono type of essence, resulting in video and audio essence data being separately stored in different MXF files. **Figure 10** on the next page schematically illustrates the MXF internal behavior for a typical example of the MXF OP Atom files containing audio and video essences separately.

In this figure, the File Packages in the MXF files, or "FP1" and "FP2", are assumed to describe the respective Essence Containers that store either video or audio essence data. While no constraint is specified to the Material Package for the MXF OP Atom files [12], the most typical use of the Material Packages in the MXF OP Atom files is demonstrated in **Figure 10**, where the Material Package in each MXF file references not only to the File Package in the same file but also to those in other files. As a result, each Material Package receives the baseband essence flow not only from the File Package in the same file but also from those in other files over the file boundaries, which are then synchronized to produce the audiovisual essence flow to output.

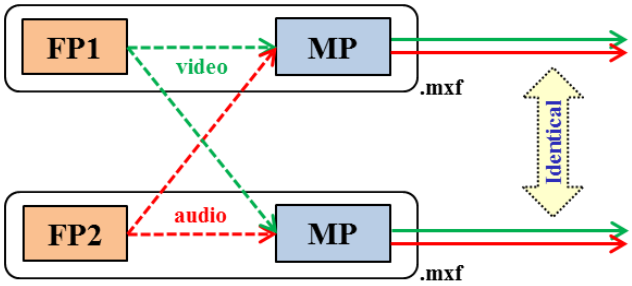


Figure 10: MXF OP Atom internal behavior model

Note that because the Material Package contained in any MXF file in the most typical MXF OP Atom files described above theoretically receives the same set of baseband essence flows to be synchronized from the File Packages, any of them can be chosen to obtain identical playout result. Compared with the AMWA AS-02 files discussed in Section 2.4.5, on the other hand, because the baseband essence flow supplied by the File Package is *imperceptive*, there is *theoretically* no way for an external application to access just a mono essence, say, either video or audio essence, when an audiovisual material is represented as the MXF OP Atom files[3].

It is also noted that the MXF OP Atom is not a part of the OPs specified by SMPTE ST 377-1 [4] but regarded as a specialized one lying outside the realm of the standard operational patterns.

2.4.7 Operational Pattern OP*ic* (*i*=1,2,3)

So far, only one Material Package is contained in a single MXF file, resulting in one kind of essence flow to be produced to output at playout. However, SMPTE ST 377-1 [4] also defies the MXF OPs where two or more kinds of essence flows are to be selectively produced to output even from a single MXF file, which is called the Alternative Package, or MXF OP*ic* (*i*=1, 2, 3) [13].

According to SMPTE ST 377-1, this is realized by letting an MXF file contain two or more Material Packages, each of which describes which set of File Packages it will receive the baseband essence flows from, and how to synchronize

[3] In reality, however, because an external application often knows which MXF file stores a desired mono essences based on, *e.g.*, their file names, it will directly access the essence as needed regardless of the description in the MXF Header Metadata.

them. **Figure 11** below schematically illustrates the MXF internal behavior for an example of the MXF OP1c file that can selectively produce two kinds of synchronized audiovisual essence flows to output.

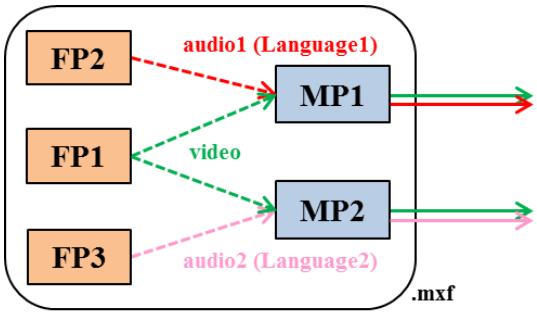


Figure 11: MXF OP1c internal behavior model

In this figure, under the assumption of an audiovisual program with two different languages (Language1 and Language2), three File Packages, “FP1”, “FP2” and “FP3”, are contained in a single MXF OP1c file to describe the Essence Containers for video essence (“video”) and audio essences of each language (“audio1” for Language1 and “audio2” for Language2), respectively. Then, there are two Material Packages contained in the file, among which one (“MP1”) references to the “FP1” (for “video”) and “FP2” (for “audio1”) to produce the audiovisual essence flow with the Language1, while another (“MP2”) references to the “FP1” (for “video”) and “FP3” (for “audio2”) for the audiovisual essence flow with the Language2.

Note that no constraint is specified on how to store the Essence Containers for video and audio essences in a file in these OP specifications, *i.e.*, they could be separately localized, or each of them could be divided into segments and multiplexed by using the Body Partition Pack as a separator for their easy synchronization in the file. Furthermore, some or all of them could be provided as external MXF files in the same way as discussed in Section 2.4.5.

It ought to be also noted that while more than one kind of playout result is obtained from a single MXF OP*i*c (*i*=1, 2, 3) file, SMPTE ST 377-1 [4] does not specify how to select a particular output in the file. Instead, it specifies the Primary Package property in the Preface (See **Figure 2**) in order to indicate the default Material Package to be used when an MXF OP*i*c (*i*=1,2,3) file is just selected for playout without any special options.

3 UMID Application Basics

3.1 Introduction

This section briefly introduces the UMID application basics that ought to be referenced as a basis of the UMID applications in MXF. After introducing the UMID format, the UMID Application Principles are reproduced, followed by two distinct functions of UMID as the basic UMID applications.

3.2 UMID Format

The UMID is a byte string of either 32 or 64 bytes. The former is called Basic UMID, a core component as a unique material identifier. The latter is called Extended UMID and is composed of two parts: 32-byte Basic UMID followed by 32-byte Source Pack, as shown in **Figure 12**.

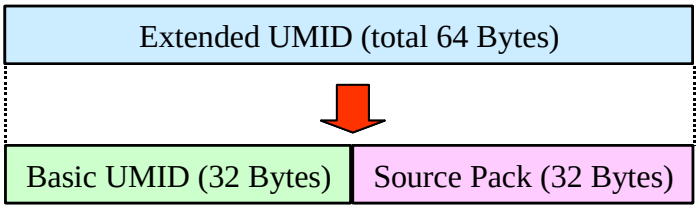


Figure 12: Top-level structure of UMID

The UMID can be used either as the 32-byte Basic UMID on its own or as the 64-byte Extended UMID.

Figure 13 below shows the Basic UMID format, which is composed of the following four fields:

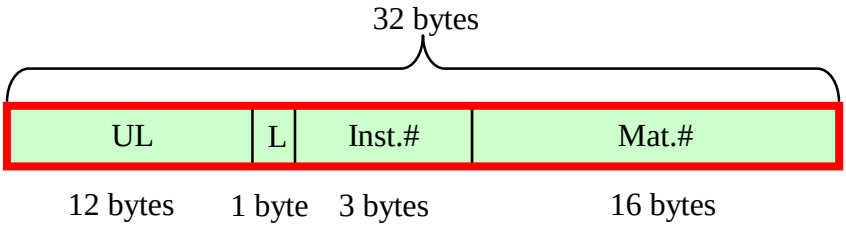


Figure 13: Basic UMID format

- **SMPTE Universal Label (UL)**: The first 12 bytes of the Basic UMID constitute the SMPTE Universal Label. The first 10 bytes are fixed values based on the registered International Organization for Standardization label administered by SMPTE. The 11th byte indicates the type of material this UMID identifies. The 12th byte is divided into top and bottom nibbles: the top nibble composed of 4 most significant bits (MSBs) and the bottom nibble of 4 least significant bits (LSBs) indicate the number creation methods for the Material Number (Mat.#) and Instance Number (Inst.#), respectively. Note that the bottom nibble is also used to indicate the "Live stream", *i.e.*, when specified as "F_h", it indicates that a material attached with this UMID is a direct live signal source from a material creation device, resulting in un-persistent and thus cannot be uniquely identified,
- **Length (L)**: This 1 byte field specifies the length of byte string that follows. Since 19 bytes follow in the case of Basic UMID, this field is fixed to 13_h, while it is 33_h for the Extended UMID because of additional 32 byte Source Pack that follows,
- **Instance Number (Inst.#)**: This 3-byte field specifies whether the Material Number, the field that immediately follows this, is a newly created value or the inherited one from another UMID already existing elsewhere. For a newly created UMID, this field must be zero-filled (00_h 00_h 00_h), indicating that the Material Number is a newly created value. While several methods are defined for the creation of non-zero Instance Number, the distinction of zero or non-zero value for it is significant for most UMID applications,
- **Material Number (Mat.#)**: This 16 byte field accommodates a globally unique value, which makes the UMID also a globally unique material identifier. Several creation methods of the value for the field are specified, among

which an example is given by the combination of the network node number of a device creating a material together with the time snap at which the material is created. Because all network node number is globally unique, a material with UMID of this combination can be also globally uniquely identified under the assumption of a single material being created at the time snap. Note that this method creates a self-identifying globally unique value without access to either the registration authority or the central database, which is in common use in the information technology system such as for UUID.

When an individual material unit, or quantum duration of a material such as a video frame or an AES3 audio frame, needs to be uniquely identified within a material, an additional component called Source Pack is appended to the Basic UMID, which forms the Extended UMID as shown in **Figure 12**.

Figure 14 below shows the Extended UMID format, where the Source Pack is composed of the following three fields:

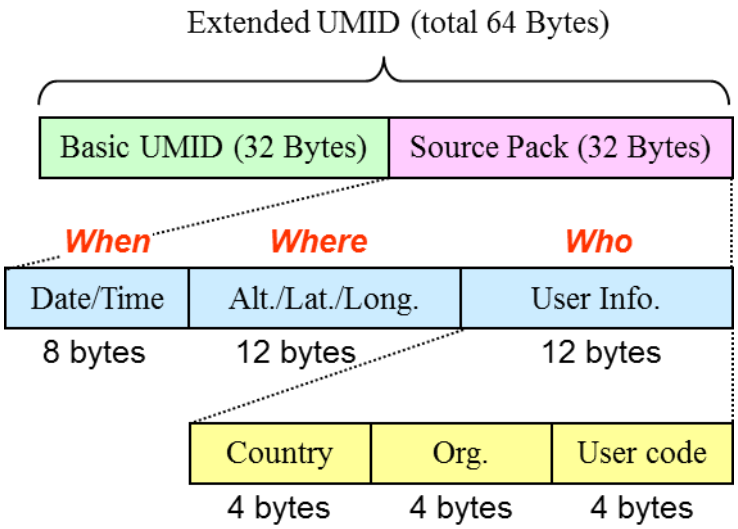


Figure 14: Extended UMID format

- **"When" field (Date/Time)**: This 8-byte field specifies the date and time stamp at which a target material unit is initially created. The timing granularity for this field needs be smaller than the cyclic period of material unit so that each material unit can be uniquely identified with the value for this field,
- **"Where" field (Alt./Lat./Long.)**: This 12-byte field specifies the spatial co-ordinate information associated with a target material unit. This field is further decomposed into three parts: Altitude, Latitude and Longitude, each of which is 4 bytes long,
- **"Who" Field (User Info)**: This 12-byte field specifies who initially created the target material unit as a string of pre-registered text. This field is further decomposed into three parts: Country, Organization and User code, each of which is 4 bytes long.

While the original intention of the Extended UMID was to identify individual material unit that comprises the material, it also accommodates fundamental information about the creation of each material unit, which is expected to preserve throughout the processes of the media production workflow chain.

3.3 UMID Application Principles

3.3.1 What are the UMID Application Principles?

The UMID Application Principles are a set of golden rules applicable to all kinds of UMID applications, which are to be specified by the latest SMPTE RP 205 [2]. By definition, any MXF products that will treat UMID in the ways discussed in this report need to strictly follow the UMID Application Principles.

In the next subsection, the UMID Application Principles are reproduced *as they are* so that they are referenced as a basis of the discussion on the UMID applications in MXF.

3.3.2 UMID Application Principles

Principle 1 – Definitions

For the purpose of describing the UMID Application Principles, the capitalized terms used in the statements below shall have the respective meanings defined as follows:

- **Basic UMID** shall mean the UMID of 32-byte long composed of its12-byte UL, the 1-byte length field, 3-byte Instance Number Field and 16-byte Material Number Field, specified in SMPTE ST 330. While this term may represent either on its own or the first 32-byte of Extended UMID, the phrase "the Basic part of Extended UMID" may be used to refer to the latter case for its further clarification if needed.
- **Extended UMID** shall mean the UMID of 64-byte long composed of its 32-byte Basic UMID followed by 32-byte Source Pack, used to identify a finer granularity than is identified by its Basic part, specified in SMPTE ST 330.
- **Instance** shall mean a specific item of stored material(s) that shares the same Material Number regardless of its Instance Number value.
- **Instance Number** shall mean the value in the Instance Number Field of the UMID. The Instance Number shall take a value of zero for the original Material and a non-zero value for any copy or derivation of the original Material.
- **Instance Number Field** shall mean the 24 bits field using bytes 14^{th} - 16^{th} of the UMID as specified in SMPTE ST 330.
- **Instance Number Generation Method Field** shall mean the field of the 4 least significant bits (LSBs) of byte 12^{th} of the UMID UL used to identify the method to create the Instance Number field value as specified in SMPTE ST 330. Note that the special value of 15 (F_h) is reserved for this field in order for the UMID with it to be used to signal a live-stream source that has never been fully recorded (and hence has never been instantiated as a persistent and deterministic form of material) rather than to be used as a unique material identifier.
- **Material** shall mean a persistent and deterministic form of unique set of audio and/or visual essence playable on a single timeline, which may be a single item or a set of synchronized items, which shall be continuous along the timeline. Material, often preceded with the term "original", shall be globally uniquely identified by a UMID with a newly created Material Number together with zero Instance Number, whose global uniqueness is algorithmically guaranteed according to SMPTE ST 330. The essence in the Material may either be a playable bit stream, or an abstract source from which a playable bit stream may be created.
- **Material Number** shall mean a non-zero value for the Material Number Field in the UMID.
- **Material Number Field** shall mean the field of bytes 17^{th} - 32^{nd} of UMID as specified in SMPTE ST 330.
- **Material Unit** shall mean the quantum of material composed as its cyclic sampling structure.
- **Source Pack** shall mean the last 32 bytes of the Extended UMID, which contains the information of "when", "where" and "who" has created the material, as specified in SMPTE ST 330.
- **UL** shall mean the SMPTE Universal Label specified in SMPTE ST 298, which is truncated to 12 bytes length when used in the UMID format. When the UMID UL is extracted for external use, it shall be padded to 16 bytes by appending 4 bytes of zero value and shall change the value of the label size in byte 2^{nd} from $0A_h$ to $0E_h$.
- **UMID** shall mean the Unique Material IDentifier whose format and value creation methods are specified in SMPTE ST 330. The UMID may take the form of either 32-byte Basic UMID or 64-byte Extended UMID, where the Extended UMID is still a UMID, but the one extended with additional 32 bytes of geo-location and other information to form the Source Pack.
- **UMID Managed Domain** shall be an authoritative source in which the uniqueness and meaning of a UMIDs Material Number and Instance Number shall be managed and guaranteed. A UMID Managed Domain may range from a single registry, within a single location, to a large federated registry spanning multiple organizations. UMID Managed Domains shall govern the Law of Identity for UMIDs within their scope of authority.

Principle 2 – UMID Creation

A UMID based material management system shall create a UMID with a newly created Material Number together with a zero Instance Number in accordance with SMPTE ST 330 for original Material at its point of entry into the UMID Managed Domain.

Principle 3 – UMID Integrity

Different original Materials shall be globally uniquely identified by different UMIDs.

Principle 4 – UMID Identification

If more than one material is uniquely identified by a single UMID, their representations at playout shall be identical bit by bit on the timeline.

Principle 5 – UMID Inheritance

Any Instance derived from an original Material may be attached with a UMID composed of Material Number inherited from that of the original Material together with the non-zero Instance Number.

Principle 6 – Extended UMID

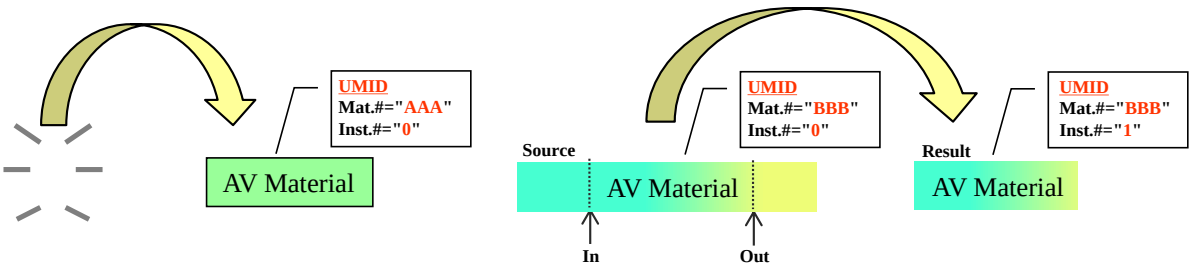
Any Extended UMID attached to a Material Unit belonging to the same material shall share the same Basic UMID that uniquely identifies the material as a whole.

Principle 7 – Source Pack

The Source Pack should identify the time, space and ownership properties of the material at its point of creation or capture. If present as a part of an Extended UMID, its values shall not be replaced with new ones when the basic part of its UMID is inherited.

3.4 Two distinct functions of UMID

According to the UMID application basics discussed in Annex C.2 of [2], there are two distinct functions of UMID, *i.e.*, a UMID as a globally unique material identifier and that as a linking tool. The former function is achieved when a globally unique value is set to the UMID Material Number (Mat.#) and zero value to the UMID Instance Number (Inst.#), while the inherited value to the Mat.# and non-zero value to the Inst.# for the latter function, as shown in **Figure 15** below.



(a) As a globally unique material identifier (b) As a linking tool

Figure 15: Two distinct functions of UMID

4 UMID Applications in MXF

4.1 Introduction

This section constitutes the main body of this study report. After introducing three kinds of UMID in an MXF file, the applications of those UMIDs in various cases of the MXF file treatment based on the UMID Application Principles are specifically discussed one by one.

4.2 UMIDs in MXF

4.2.1 MXF Package UIDs

According to SMPTE ST 377-1 [4], the Basic UMID of 32-byte long is used as a unique identifier (UID) of the Package that describes the essence on a timeline. As discussed in Section 2.3, there are two kinds of Packages: the Material Package to describe the essence on the output timeline at the playout of an MXF file and the File Package(s) to describe the essence(s) stored in the Essence Container(s) along with the input timeline(s).

While the Package UID is introduced to uniquely identify a Package instance within a file by definition, it can be also regarded to uniquely identify an essence flow supplied by the Package identified by the UID, *i.e.*, the *imperceptive* baseband essence flow by the File Package UID and the essence flow to output by the Material Package UID, based on the MXF internal behavior model discussed in Section 2.3. **Figure 16** below shows the MXF internal behavior model with the Package UIDs being attached to the respective Packages.

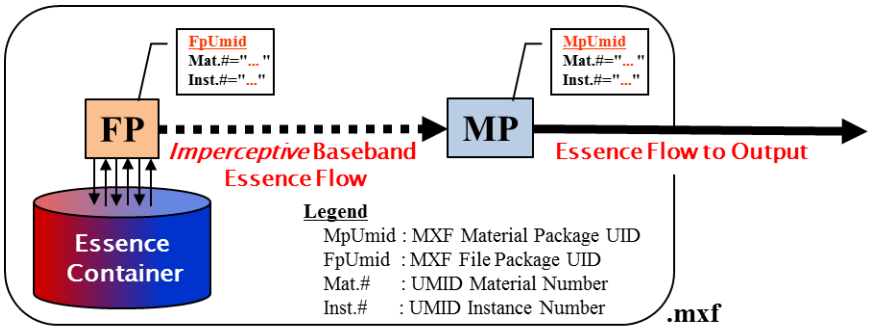


Figure 16: MXF Package UIDs (MpUmid and FpUmid)

In the following discussion, let MpUmid and FpUmid denote the Package UIDs of Material and File Packages, respectively. The notations of Mat.# and Inst.#, which are introduced in Section 3.2 to represent the UMID Material Number of 16-byte long and the UMID Instance Number of 3-byte long, respectively, are also used as depicted in **Figure 16**.

4.2.2 Body UMID

In addition to the MXF Package UIDs, there is another kind of UMID in an MXF file called Body UMID. Specifically, it is either the Basic UMID of 32-byte long or the Extended UMID of 64-byte long associated with a frame (or a group of frames if desired) and inserted into the MXF File Body via so called Generic Container [14, 15] as the Essence Container. **Figure 17** on the next page schematically illustrates an example of the detailed structure of the Generic Container containing the Body UMIDs.

As shown in the figure, the File Body is composed of a sequence of Edit Units, which is per frame basis in this case. An Edit Unit for a frame is then composed of the System Item, the Picture Item containing video data par frame, the Sound Items containing audio data synchronized with the frame, and finally the Data Item for the frame

Following the convention of SDTI-CP (Serial Data Transfer Interface – Content Package) [16], non-audiovisual data such as metadata associated with a frame can be stored either in the System Item or in the Data Item [17]. In the case of the Body UMID, it is to be stored in the System Item with the Type value of 83_h according to the relevant SMPTE standard [18].

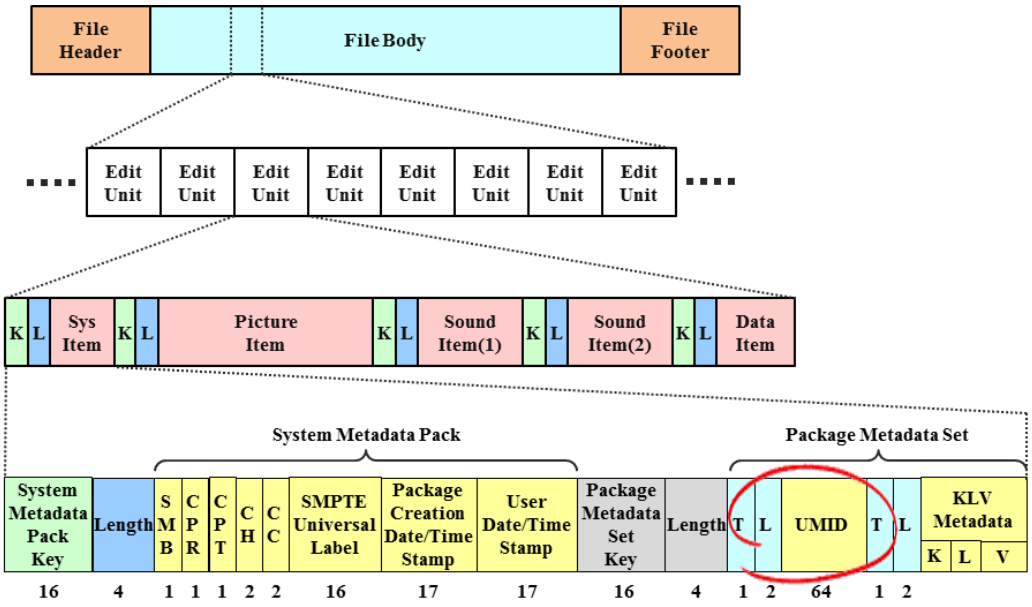


Figure 17: MXF Generic Container in file body

In the following discussion, let BodyUmid denote the Body UMID.

4.3 Applications of Material Package UID (MpUmid)

4.3.1 What does the MpUmid identify?

While the originally expected role of the MpUmid would be a unique identifier of an instance of the Material Package within an MXF file, when the functionality of Material Package to describe the essence on the output timeline at the playout of an MXF file is taken into account, the MpUmid is considered also to uniquely identify the essence flow to be produced to output, *i.e.*, what we will actually observe when the MXF file is to be played out, as discussed in Section 4.2.1. Because it is a representation of a material at its playout which a UMID uniquely identifies according to the UMID Application Principle 4 (UMID Identification), the MpUmid is regarded to uniquely identify a representation of the material at its playout, which is recorded as a form of the MXF file.

4.3.2 MpUmid as a globally unique material identifier

4.3.2.1 UMID Managed Domain

When the MpUmid is used as a globally unique material identifier, the MXF files with the MpUmid need to constitute the UMID Managed Domain as specified in Section 5.2 of [2].

The UMID Managed Domain is a conceptual domain composed of audiovisual materials with a valid UMID in the sense of the UMID Application Principles 2 to 4 (UMID Creation, Integrity and Identification), which are applied to the UMID as a globally unique material identifier. This is in fact an embodiment of those UMID Application Principles. In order for the MpUmid in the UMID Managed Domain to be always maintained valid in the sense of those UMID Application Principles, certain MpUmid treatments are usually required at every manipulation of an MXF file in the domain, which are executed by the Material Manger.

The Material Manager is a tool that is responsible to maintain the integrity of MpUmid in the UMID Managed Domain. In addition, because an MpUmid by itself does not tell anything on where to access a desired MXF file identified by the MpUmid, the Material Manger is also expected to maintain the correspondence between the MpUmid and the access information of a desired MXF file identified by the MpUmid such as its URL by using their mapping list called UMID Managed List.

4.3.2.2 Behaviors of Material Manager

4.3.2.2.1 Introduction

This subsection describes the expected behaviors of the Material Manager that maintains the integrity of MpUmid in the UMID Managed Domain by making the generic discussion on the Material Manager provided in Annex B of [2] specific to the MXF technology.

Before entering the details, high level requirements of the Material Manager for the UMID as a globally unique material identifier, which are specified in Section 5.2.2 of [2], are reproduced for reference.

- The Material Manager shall provide a globally unique UMID Material Number in accordance with SMPTE ST 330 upon demand,
- The Material Manger shall maintain the global uniqueness of UMID with zero Instance Number,
- The Material Manager shall implement the UMID Resolution Protocol, providing with zero or more information to access a material identified by a UMID in question.

It ought to be noted that because only the MpUmid as a globally unique material identifier is considered in this subsection, the Material Number (Mat.#) and the Instance Number (Inst.#) in a given MpUmid are always set to a globally unique value and zero-filled, respectively.

The following describes the expected behaviors of the Material Manager for each manipulation of an MXF file under the assumption that a material is recorded as an MXF file, and the Material Manager manages its URL together with its corresponding MpUmid globally uniquely identifying the MXF file by using the UMID Managed List.

4.3.2.2.2 New MXF file creation in the UMID Managed Domain

When a new MXF file is created from scratch by, *e.g.*, acquisition, in the domain, the Material Manager will assign a newly created MpUmid to the MXF file, and also register the pair of the MpUmid and the file's URL to the UMID Managed List.

Note that if the MXF file is accessible even during at its creation (the growing MXF file), *a.k.a.* the read-while-write operation, the MpUmid attached to the MXF file will signal the "Live stream" by setting the UMID "Live stream" flag (the bottom nibble of the byte 12th of MpUmid being set to "F$_h$"). Please visit Section 4.3.4 for further discussion of the growing MXF file and the UMID "Live stream" flag.

4.3.2.2.3 Existing MXF file deletion in the UMID Managed Domain

When an existing MXF file is deleted in the domain, the Material Manager will retain a URL of the file being deleted, detect the MpUmid and URL pair in the UMID Managed List according to the URL as a key, and un-register it from the UMID Managed List before the file deletion. Note that while there might exist more than one clone of an MXF file in the domain which shares the same MpUmid as that of the MXF file being deleted, the use of the URL as a key can unambiguously identify the MpUmid and URL pair to be un-registered from the List.

It ought to be noted that the un-registration of the MpUmid and URL pair from the UMID Managed List eventually results in its deletion from the viewpoint of an external application that will request the Material Manager for the MpUmid resolution regardless whether the MXF file is actually deleted or not.

4.3.2.2.4 Existing MXF file copy or move within the UMID Managed Domain

When a new MXF file is created by copying an existing source MXF file within the domain, the Material Manager will detect the MpUmid of a source MXF file, assign it also to the copied MXF file, and register the pair of the MpUmid and the URL of a copied file to the UMID Managed List[4].

If this is a move operation where the source MXF file in the domain is deleted after the copy operation, the behavior for the existing NXF file deletion will apply to the source MXF file[5].

[4] In practice, this behavior can be reduced to duplicating the MpUmid and URL pair of the source MXF file within the UMID Managed List and updating the URL of one of the pairs to the value of a copied MXF file after the file copy operation.

Note that in either case, the Material Manager might intentionally replace the MpUmid of resulting MXF file with a newly created value for some reasons, which is still valid in the sense of the UMID Application Principles.

It ought to be also noted that if the copy operation fails and results in leaving an incomplete MXF file, the resulting MXF file must be assigned with a newly created MpUmid to unambiguously distinguish it from its source MXF file. Assuming such an incomplete MXF file is still to be used rather than to be deleted for the rollback, a practical behavior the Material Manager will take is to tentatively assign a newly created MpUmid (or the MpUmid with the UMID "Live stream" flag when the read-while-write operation is permitted) to the resulting MXF file at the beginning of the copy or move operation, and then to replace the tentative MpUmid with that of its source MXF file at the successful completion of the operation.

4.3.2.2.5 New MXF file import into the UMID Managed Domain

When a new MXF file is imported to the domain by copying or moving from elsewhere, the default behavior the Material Manager will take is to detect the MpUmid of an incoming MXF file, to replace it with a newly created UMID value if detected (or to simply assign a newly created MpUmid to the MXF file if not detected), and to register the pair of the MpUmid and the resulting file's URL in the domain to the UMID Managed List.

Note that this is also applicable to an MXF file existing on the same physical medium or elsewhere but not recognized by the Material Manager when it is to be imported into the UMID Managed Domain.

It ought to be also noted that this implies that the value of the original MpUmid is lost in the import operation. In the case of MXF, however, thanks to the dual layered structure of the MXF Packages discussed in Section 2.3.1, there is a way for such an original MpUmid value to be retained even for the imported material as an MXF file. Please see Section 4.4.3 for further discussion for it.

An exceptional behavior can be applied when the location from which an MXF file is to be imported is also known as the UMID Managed Domain. In this case, because the MpUmid attached to the incoming MXF file is valid in the sense of the UMID Application Principles by definition, the MpUmid can be reused when an MXF file is imported into the domain exactly "as is" (as a clone of the MXF file). Therefore the Material Manager in this case will just detect the MpUmid of an incoming MXF file, and then register the pair of the MpUmid and the resulting file's URL in the domain to the UMID Managed List[6].

Care needs to be taken that this behavior is effective only when the detected MpUmid is deemed as a valid globally unique material identifier with a zero Inst.# value. In other words, if the detected MpUmid has a non-zero Inst.# value or the UMID "Live stream" flag (the bottom nibble of the byte 12th of UMID being set to "F_h") resulted from copying a source MXF file which is also still under recording, the default behavior, or to replace it with a newly created UMID value, will apply because of its invalidity as a globally unique material identifier.

When an MXF file with a valid MpUmid is imported from the UMID Managed Domain, on the other hand, an MXF file being recorded in the domain will signal its status by setting the UMID "Live stream" flag to "F_h" if the MXF file is made accessible before completion of its recording. As a result, even if the MXF file import is discontinued before the completion for some reasons, the incompleteness of the remaining MXF file in the domain can be detected by its MpUmid with the flag, which needs to be replaced with a newly created UMID value when the incompletely recorded MXF file is still to be used in the UMID Managed Domain.

It ought to be also noted that, even though an MXF file is known to come from another UMID Managed Domain in advance, the Material Manager might intentionally replace the incoming MpUmid with a newly created UMID value for some reasons, which is still valid in the sense of the UMID Application Principles.

---

[5] If the operation is known as the move in advance, a practical behavior of the Material Manager will be just to update the URL value to that of destination of the MXF file being moved.

[6] According to the UMID Application Principle 4 (UMID Identification), any identical MXF files can share the same MpUmid. Consequently, this behavior is applicable to the MXF file import not only as a clone but also as an identical one in the sense of the Principle such as just for its file wrapper or its descriptive metadata being changed including the MXF OP changes, *e.g.*, from the MXF Atom files to the MXF OP1a file while its essence kept unchanged.

4.3.2.2.6 Existing MXF file export from the UMID Managed Domain

When an existing MXF file in the domain is exported to outside of the domain, no particular behavior is required by the Material Manager as long as the source MXF file remains as is in the domain. It is the responsibility of the Material Manager in destination (if exists) to guarantee the validity of the MpUmid. If this is a move operation where the source MXF file in the domain is deleted after the export, the behavior for the existing MXF file deletion will apply to the source MXF file.

Note that if the destination of an MXF file export is not the UMID Managed Domain, the exported MXF file is thus no more managed according to its MpUmid. As a result, the MXF file at the destination cannot be seen from an external application that will request the Material Manager for its MpUmid resolution to its corresponding URL.

4.3.2.2.7 Existing MXF file modification at its essence in the UMID Managed Domain

When an existing MXF file in the domain is modified at its essence by, for example, insert editing, the Material Manager will find the MpUmid of the MXF file, replace it with a newly created UMID value, and update the MpUmid value in the pair of the MpUmid and the file's URL existing in the UMID Managed List.

This needs to be conducted even when only one MXF file is known to exist for a given MpUmid within the domain, *i.e.*, because the scope of uniqueness of MpUmid is global, we always need to assume the existence of identical MXF files sharing the same MpUmid elsewhere unknown. Hence, in order to meet the UMID Application Principle 3 (UMID Integrity), it is mandatory for the Material Manager to update the MpUmid at any essence modification of an MXF file except for the essence manipulations that logically and fully preserve the identicalness of the MXF file such as mathematically lossless compression or the MXF OP changes[7].

It ought to be noted that any MpUmid update at its essence modification will break the link to an MXF file from any external entity elsewhere, such as external metadata. Hence it is an application's responsibility for the resulting MXF file to be re-linked to the external entity if needed such as to the external metadata irrelevant to the technical properties of an essence.

In terms of non-audiovisual data contained in and/or externally referenced from an MXF file, whether it is regarded as an essence or the descriptive metadata determines the treatment of the MpUmid of the MXF file. For example, when a subtitle referenced from an MXF file for its overlay at playout is modified, the MpUmid of the MXF file needs to be updated when the subtitle is regarded as a sort of essence[8]. Otherwise (so regarded as the desctiptive metadata), the MpUmid of the MXF file can remain as is.

4.3.2.2.8 Existing MXF file modification at its metadata in the UMID Managed Domain

When an existing MXF file in the domain is modified at its metadata, the behavior the Material Manager will take depends on the types of metadata being modified.

If the metadata is the MXF Descriptive Metadata such as title or content synopsis, there is nothing for the Material Manager to take because it does not affect the representation of an MXF file at its playout.

If the metadata is the MXF Structural Metadata based on which the representation of an MXF file at its playout is controlled, the Material Manager will usually detect and update the MpUmid to a newly created UMID value in order to meet the UMID Application Principle 3 (UMID Integrity).

For example, if the Material Package in an MXF file is modified, it usually requires the MpUmid being updated because of its functionality to describe the essence on the output timeline at the playout of an MXF file. The only exception is a case that the Material Package modification does not affect the essence flow to be produced to output

[7] Although no action seems needed at the MXF file modification to meet the UMID Application Principle 3 (UMID Integrity) if the nonexistence of an identical MXF file elsewhere is logically proven, it does not meet the UMID Application Principle 4 (UMID Identification) at all. This would be better understood by such an extreme example that when the essence in an MXF file is completely replaced with another essence, it will make sense for its MpUmid to be also replaced with a newly created UMID value.

[8] When the subtitle as an essence is modified, it is likely that the unique material identifier attached to the subtitle is also updated, resulting in that an application of the MXF file will notice it by detecting the un-link to the original subtitle. This is not happening when the URL is used for the subtitle linking, *i.e.*, if the URL remains unchanged even after the subtitle modification, the materials with different subtitles would have shared the same MpUmid, which breaches the UMID Application Principle 3 (UMID Integrity).

at all. For example, the MpUmid needs not to be updated when the start timecode is changed in the Material Package. This MpUmid treatment would differentiate the role of MpUmid from that of originally expected one as a unique identifier of an instance of the Material Package within a MXF file, which is supposed to be updated at any instance modification.

4.3.2.2.9 Existing MXF file backup and restore

When an MXF file existing in the UMID Managed Domain is stored into another storage medium for backup, the Material Manager will take no action even though the MXF file in the storage medium becomes out of scope of the Manager. This is because such a storage medium will be regarded as an extended area of the domain directly managed by the Material Manager, where MXF files in the storage medium are considered equivalent with those in the domain and thus their MpUmid Integrity is guaranteed.

Hence, when a backed up MXF file in the storage medium is to be restored to the original domain, the Material Manager will take the same behavior as for the import from another UMID Managed Domain.

It ought to be noted that this is based on the assumption that the backup is regarded as an operation to store and to restore a clone of an MXF file, and therefore, if only a part of the backed up MXF file is to be restored to the domain by so called partial retrieval operation, this assumption does not hold and the behavior for the import from somewhere unknown will apply.

4.3.2.2.10 MXF file managed by plural Material Managers

Theoretically, a single MXF file can be managed by more than one Material Manager. Because the MpUmid attached to an MXF file in the UMID Managed Domain is valid by definition, another Material Manager can take such an action that it does not actually import the MXF file into its own UMID Managed Domain but just references to the MXF file and registers the MpUmid and URL pair for the MXF file to its own UMID Managed List.

Because each Material Manager basically functions independently, a problem occurs when a Material Manager is involved in an MXF file manipulation that will change the state of the MXF file. For example, when one Material Manager is involved in an MXF file deletion, the Manager would un-register the MpUmid and URL pair for the MXF file being deleted from its own UMID Managed List. But because the pair in the UMID Managed List owned by another Material Manager remains as is unless appropriately notified with the deletion, the other Manager will respond to an external application with the URL which is no more effective if the MpUmid resolution is requested.

More serious problem occurs when a Material Manager is involved in a modification of an MXF file at its essence. Although one Material Manager involved in the modification of an MXF file will update the MpUmid attached to the MXF file, it could happen that the MpUmid update is not reflected to the UMID Managed List owned by other Material Manager in a timely manner, resulting in that the other Manager continues to resolve the old MpUmid to the URL at which the original MXF file identified by the old MpUmid does no more exist but modified one with the updated MpUmid.

Ideally, the Material Manager is expected to periodically examine the validity of MpUmid in its UMID Managed List by inspecting all the MpUmids attached to the MXF files in the domain the Manager manages. In reality, however, it would be too cumbersome for the Manager to always check the actual MpUmids for the MXF files.

To address this problem, one possible solution will be to compare the time stamp attached to the MXF file at its modification (described by the Identification shown in **Figure 2**) with that attached to the MpUmid and URL pair at its registration to the UMID Managed List.

If we make it a rule to register the MpUmid and URL pair to the UMID Managed List *after* the completion of the MXF file modification, the time stamp for the MpUmid and URL pair registration is always newer than that for the MXF file modification. Consequently, if a Material Manager detects the former time stamp being older than the latter, the MpUmid in the UMID Managed List of the Manager might be obsolete because it indicates that another MXF file modification occurs after the registration of the MpUmid and URL pair to the UMID Managed List, which might change the state of the MXF file and thus update the MpUmid.

This problem happens only for the MXF file manipulations that will change the state of the MXF file and thus the MpUmid such as the MXF file modification at its essence. Hence, we can avoid the problem if we prohibit such manipulations but just permit the read-only operation that will never change the state of the MXF file.

Therefore, if you want to manage MXF files that might exist in other UMID Managed Domains (such as those in another independent storage device), it would be better for you to treat it as a read-only MXF file, *i.e.*, you ought not to delete or even slightly modify the essence in the MXF file in order to avoid inconsistency in the UMID Managed List owned by other Material Managers.

4.3.2.3 Comparison of the MpUmid and a unique identifier of an MXF file

While the MpUmid could be regarded as a unique identifier of an MXF file in many cases, this is not always true. As discussed in Section 4.3.1, what the MpUmid uniquely identifies is not the file itself but the essence flow to be produced to output at the playout of an MXF file. This discrepancy can be seen when two or more representations are made possible by a single MXF file at its playout, *i.e.*, in the case of the Alternate Package or MXF OP*i*c (*i*=1, 2, 3) file.

According to Section 2.4.7, the MXF OP*i*c (*i*=1, 2, 3) file contains two or more Material Packages in order to support multiple representations at the playout of a single MXF file. Therefore, an MXF OP*i*c (*i*=1, 2, 3) file by itself can be identified by two or more MpUmids, each of which identify a particular representation of the MXF file at its playout, resulting in multiple MpUmid and URL pairs that shares a single URL being registered to the UMID Managed List.

It ought to be noted that since the Primary Package property in the Preface (See **Figure 2**) indicates the default Material Package to be used when an MXF OP*i*c (*i*=1, 2, 3) file is just selected for playout, the MpUmid attached to the default Material Package can be regarded to be a representative of the MXF file in general, *i.e.*, as an unique identifier of the MXF file, if needed.

4.3.3 MpUmid as a linking tool

There are cases where the MpUmid is used as a linking tool to associate an MXF file identified by the MpUmid with any other external entities. The typical examples include a high-resolution original material and its low-resolution lightweight proxy material, both of which are simultaneously created at acquisition and recorded as MXF files, as schematically illustrated in **Figure 18** below.

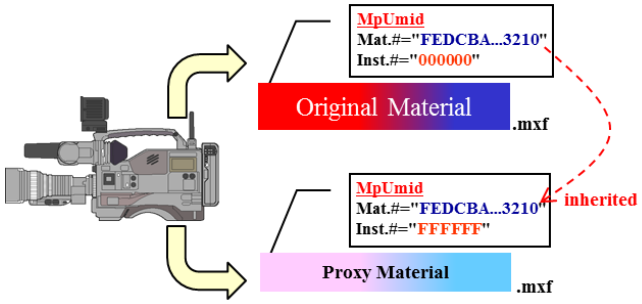


Figure 18: Simultaneously created original and proxy MXF files

As show in the figure, the MXF file for the original material, or the original MXF file, is assigned with the MpUmid as a globally unique material identifier that will be appropriately managed in the UMID Managed Domain as discussed in Section 4.3.2. The MXF file for the proxy, or the proxy MXF file, has the MpUmid as a linking tool, whose Material Number (Mat.#) is inherited from that of the original MXF file while the Instance Number (Inst.#) is set to non-zero value("FF_h FF_h FF_h" in this case) so that the proxy MXF file is unambiguously associated with the original one.

Because the original MXF file is usually huge in its data size, the proxy MXF file is often used for its content viewing and/or so called offline edit instead of the original one. Annex C.3 of [2] demonstrates how easily the EDL (Edit Decision List) obtained through the offline edit of the proxy MXF files can be translated into the EDL to be applied to the original MXF files.

It ought to be noted that the MpUmid used as a linking tool cannot be a globally unique material identifier, *i.e.*, its uniqueness is guaranteed only within a predefined closed domain where the uniqueness of the Inst.# needs to be appropriately controlled by using such as the local registration. Therefore, the proxy MXF file assigned with such an MpUmid is no more regarded as an independent material to constitute the UMID Managed Domain discussed in Section 4.3.2.1 but as a sort of metadata associated with the original MXF file. In other words, if the proxy MXF file is also desired to be managed in the same way as for the original MXF file, its MpUmid needs to be replaced with a

newly created UMID as a globally unique material identifier and then registered with its corresponding URL to the UMID Managed List owned by the Material Manager.

4.3.4 MpUmid for the growing MXF file

If an MXF file is accessible even during at its creation (the growing MXF file), *a.k.a.* the read-while-write operation, the MpUmid attached to such a glowing MXF file cannot be a unique material identifier in principle because it breaches the UMID Application Principle 3 (UMID Integrity), *i.e.*, as its duration continuously changes during growing, it is not deterministic and thus cannot be said unique[9].

On the other hand, a simple mechanism has been often desired to signal that a particular MXF file is under growing, for which the MpUmid with the UMID "Live stream" flag can be utilized.

The UMID "Live stream" flag is the value of "F_h" that is set to the bottom nibble of the byte 12th of UMID (the field for Instance Number Generation Method). According to SMPTE ST 330 [1], the UMID "Live stream" flag, or the value of "F_h" for the field, is specified as

> "*This instance number method shall be used to identify that the material is a direct live signal source from a material creation device*",

and, therefore, the MpUmid with the UMID "Live stream" flag can be used to simply signal that an MXF file attached with it is under growing rather than as a unique material identifier.

When recording stops, resulting in the MXF file becoming persistent and thus can be globally uniquely identified, its MpUmid needs to be made valid by replacing it with a newly created UMID value with zero Inst.#.

The registration of the MpUmid and URL pair to the UMID Managed List is required only when the MXF file becomes persistent with a newly created MpUmid as its valid globally unique material identifier.

It ought to be noted that the use of UMID "Live stream" flag for a growing MXF file is just as a simple tool to signal the file growing, where only the value of "F_h" at the field for Instance Number Generation Method is significant in principle regardless of the value specified at the rest of the fields in the MpUmid (except for the case in which its Mat.# is used to uniquely identify the material "source". See Section 4.4.3.5 for more discussion). Because this use of the MpUmid is not as a globally unique material identifier, it needs to be replaced with a newly created UMID value with zero Inst.# at the completion of the MXF file growing as mentioned above and then registered with its corresponding URL to the UMID Managed List owned by the Material Manager.

4.4 Applications of File Package UID (FpUmid)

4.4.1 What does the FpUmid identify?

While the originally expected role of the FpUmid would be a unique identifier of each instance of the File Package within an MXF file, when the functionality of File Package(s) to describe the essence(s) stored in the Essence Container(s) along with the input timeline(s) is taken into account, the FpUmid is considered also to uniquely identify an *imperceptive* baseband essence flow supplied by the File Package with the FpUmid to the Material Package as discussed in Section 4.2.1.

Unlike the MpUmid, however, the usefulness of FpUmid is insignificant for an external application because the baseband essence flow supplied by the File Package is just a conceptual entity effective only within the MXF internal behavior model discussed in Section 2.3.2. In other words, an external application cannot access it even when a given FpUmid is resolved to an MXF file that contains the File Package identified by the FpUmid. Hence, the FpUmid management for an external application in such a way as for the MpUmid discussed in Section 4.3.2 does not pay[10].

---

[9] There is an experimental implementation of a growing MXF file whose snapshot is frozen periodically and made accessible with an MpUmid newly created at every snapshot. Theoretically, it is valid because each frozen snapshot is deterministic and thus can be uniquely identified by a newly created MpUmid specifically for the snapshot.

[10] Theoretically, the FpUmid resolution might be useful for those which specify the File Package as the Primary Package such as for the MXF OP Atom file or the Essence Component file in AMWA AS-02. In reality, however, the discovery of a desired MXF file

On the other hand, the uniqueness of FpUmid within an MXF file (for *e.g.*, the MXF OP1a file) or among relevant MXF files (for the MXF OP Atom or AMWA AS-02 files) is crucial because the Material Package needs to unambiguously reference to the relevant File Package(s) by specifying its FpUmid (via the Source Clip), which can be easily realized by an application that creates the MXF file(s), though.

It ought to be noted that a modification of the descriptive metadata contained in the File Package instance such as a start timecode does not require the FpUmid to be updated. This is justified by the FpUmid uniquely identifying the essence flow supplied by the File Package rather than that as a unique identifier of an instance of the File Package, which would have been changed at any modification of the instance including the descriptive metadata.

When the baseband essence flow supplied by the File Package to the Material Package is produced to output *exactly as is* by the Material Package as observed in the case of an MXF OP1a file, both the FpUmid and MpUmid could be the same value because of the identicalness of what they identify, *i.e.,*, the baseband essence flows, and theoretically, it is valid because those instances are still distinct due to their different classes indicated by their ULs even if they share the same UMID value for their Package UIDs. In reality, however, it ought to be better for them to be assigned with completely different UMID values because most external applications are expected to watch only the Package UID value, rather than to watch it together with the Package UL, in order to trace the link between the Packages.

4.4.2 FpUmid as a globally unique identifier

A default treatment of the FpUmid will be to assign a newly created UMID value for it because the uniqueness of the FpUmid is vital. Thanks to the UMID creation algorithm specified in [1], a newly created FpUmid is always guaranteed to be globally unique even when it is locally created without reference to such as a central database. In this case, however, the global usefulness of FpUmid is insignificant, *i.e.*, it is sufficient for the FpUmid to be unique within an MXF file (for *e.g.*, the MXF OP1a file) or among relevant MXF files (for the MXF OP Atom or AMWA AS-02 files) so that it is used only as a hook to be referenced from the Material Package without ambiguity, though this is the most fundamental requirement the FpUmid must fully satisfy.

4.4.3 FpUmid as a linking tool

4.4.3.1 For the source MXF file

As discussed in Section 4.4.1, it is sufficient for the scope of uniqueness of the FpUmid to be within an MXF file (for *e.g.*, the MXF OP1a file) or among relevant MXF files (for the MXF OP Atom or AMWA AS-02 files) according to the requirement of FpUmid. Taking advantage of this fact proactively, the usefulness of FpUmid is further enhanced by using it as a linking tool rather than as a globally unique material identifier.

Suppose, for example, the case of partial retrieval of an MXF OP1a file, where a resulting MXF file called “Result.mxf” is obtained by partially retrieving a source MXF file called “Source.mxf” with its In/Out points being specified in a certain way as shown in **Figure 19**.

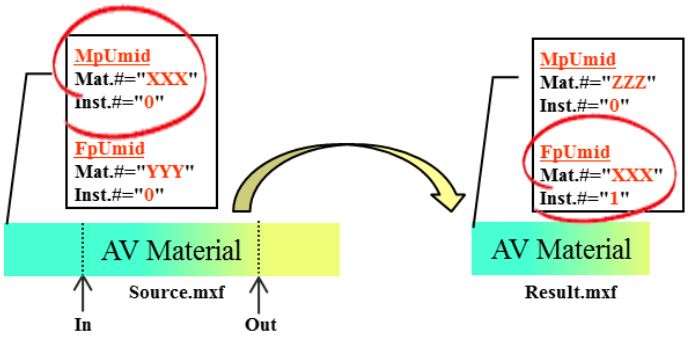


Figure 19: FpUmid application for partial retrieval of an MXF OP1a file

is more easily achieved according to its file name or by using a special file describing where a relevant file locates. See Section 4.4.3.2 for more discussion.

Because “Result.mxf” in **Figure 19** is a newly created MXF file, its MpUmid must be a newly created UMID value with its Inst.# being zero. When its FpUmid is used as a linking tool, on the other hand, its UMID value is created based on the MpUmid of “Source.mxf”, *i.e.*, the Mat.# of the FpUmid inherits the value “XXX” from the MpUmid of “Source.mxf” while the Inst.# of the FpUmid is set to non-zero value (“1”)[11] in the case of **Figure 19**.

This kind of FpUmid treatment brings additional usefulness of the FpUmid. As it is obvious in the figure, the resolution of the FpUmid of resulting “Result.mxf” with its Inst.# masked to zero will lead to the URL of “Source.mxf” MXF file. In other words, thanks to the FpUmid as a linking tool, the source MXF file from which the resulting MXF file is created can be easily obtained by applying the standard UMID resolution method to the FpUmid with its Inst.# masked to zero.

Note that while the partial retrieval is taken for example in this case, this use of the FpUmid is applicable to any kind of media processing that treats MXF file as input and output such as transcoding. Furthermore, the same approach is also applicable to any type of MXF OP files including those having multiple File Packages. **Figure 20** schematically demonstrates the case when a resulting MXF file having two File Packages (“Result.mxf”) is created from two source MXF files (“Source1.mxf” and “Source2.mxf”), where the FpUmid of each File Package in the resulting MXF file is created based on the MpUmid of respective source MXF file.

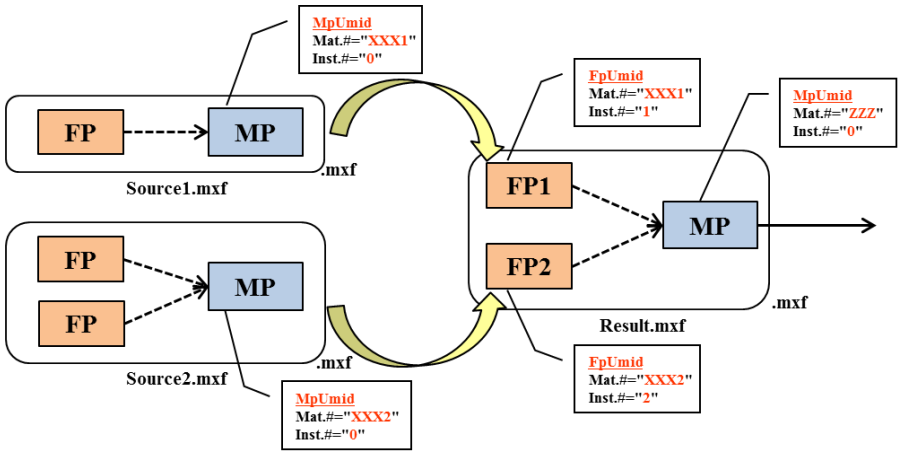


Figure 20: FpUmid application linking back to the source MXF files

As shown in the figure, the resolution of any FpUmid in the resulting MXF file with its Inst.# masked to zero will lead to the respective source MXF file from which the File Package as a part of the resulting MXF file is created[12].

When two or more File Packages are created from a single source MXF file in the case such as the cut and splice from a single source MXF file, the FpUmids attached to the File Packages will result in sharing a single value for their Mat.#. Since the FpUmid must be unique within an MXF file, those FpUmids need to be made distinct by assigning different values to their Inst.#, which, however, can be easily realized by an application that produces the resulting MXF file.

It ought to be noted that this FpUmid application is effective only when the resulting MXF file contains File Packages of the same number as the number of source MXF files (except for the case where a single source MXF file is to be used twice or more). In other words, this is not applicable when an MXF OP1a file is created from two source MXF files because the resulting MXF file contains only a single File Package by definition.

---

[11] In an extreme case where a full essence is retrieved from the “Source.mxf” to create the “Result.mxf”, the Inst.# in the FpUmid can theoretically remain zero because the baseband essence flow the FpUmid identifies is identical with what we would observe at the playout of the “Source.mxf”.

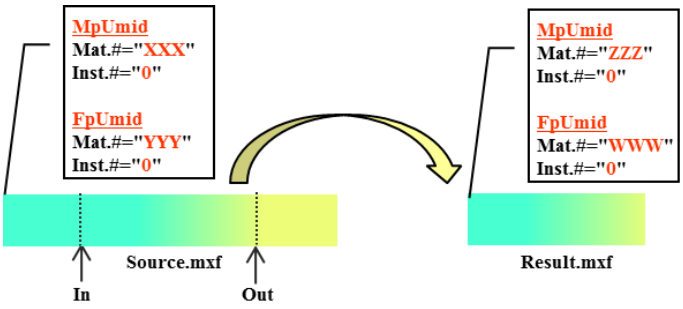
[12] A problem occurs when the FpUmid is resolved to a source material represented in the form of MXF OP Atom files sharing a single MpUmid in spite of their containing different types of essence (See Section 2.4.6). Because there is no standard method for an external application to determine which is the actual source file containing a desired essence until looking into the inside of the files, it needs to be addressed by the UMID Resolution Protocol that returns not only the URL but also the type of essence actually contained in the resolved MXF file as a part of the fundamental technical metadata.

If the number of File Packages to be created is less than the number of source MXF files, *e.g.*, in the case of an MXF OP1a file, then how to create the FpUmid depends on the consideration of relationship between source and resulting MXF files, *i.e.*, while one would choose the default behaviour to create a new UMID value for the FpUmid as discussed in Section 4.4.2, another might choose the FpUmid creation based on the MpUmid of one of the source MXF files if it is considered as dominant for the resulting MXF file creation.

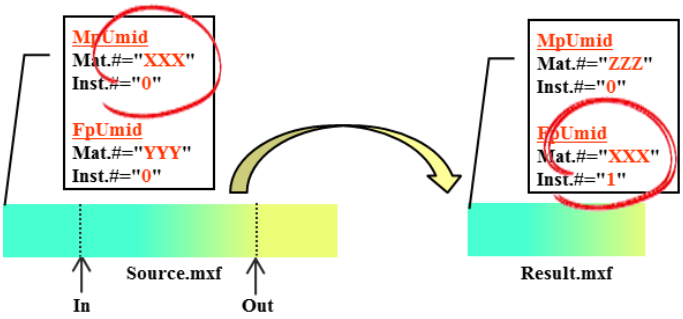
4.4.3.2 Comparison of three different FpUmid creations

While Section 4.4.3.1 discusses the FpUmid creation based on the MpUmid of a source MXF file, there are two other FpUmid creation methods. Taking the partial retrieval of an MXF OP1a file for example again, where a resulting MXF file ("Result.mxf") is obtained by partially retrieving a source MXF file ("Source.mxf"), there exist three possible FpUmid creation methods as shown in **Figure 21**, none of which is theoretically ruled out.

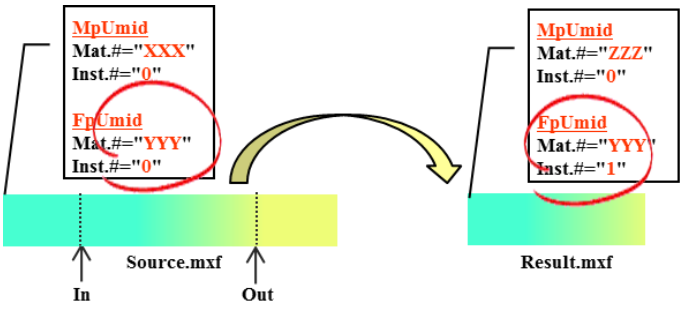
Specifically, the FpUmid in the resulting "Result.mxf" file can take a UMID of either 1) a newly created globally unique value (**Figure 21** (a)), 2) a value created based on the MpUmid of a source "Source.mxf" file (**Figure 21** (b)), or 3) a value created based on the FpUmid of a source "Source.mxf" file (**Figure 21** (c)).



(a) No reference to Source.mxf



(b) Reference to Material Package in Source.mxf



(c) Reference to File Package in Source.mxf

Figure 21: Comparison of three possible FpUmid creations

Having closer look at those options, the following considerations are made:

- The **Figure 21** (a) option has no additional function than as a unique identifier of the File Package while this is deemed a default behavior of the FpUmid creation as discussed in Section 4.4.2,
- The **Figure 21** (b) option brings additional function to reference back to the source MXF file via the resolution of the FpUmid with its Inst.# masked to zero as discussed in 4.4.3.1,
- The **Figure 21** (c) option brings additional function to reference back to the File Package of the source MXF file.

Note that although the FpUmid resolution, or a method to resolves a given FpUmid into the URL of an MXF file containing the File Package identified by the FpUmid, might be useful for some applications treating MXF files that specify the File Package as the Primary Package such as the MXF OP Atom file or the Essence Component file in AMWA AS-02, it is useless for other applications in general because they cannot access the baseband essence flow supplied by the File Package as discussed in 4.4.1.

Furthermore, because the discovery of a target MXF file in the applications desiring the FpUmid resolution can be easily achieved by using its file name or a special manifest file describing where the relevant file locates and so has been done accordingly so far, it is unlikely to happen that the FpUmid resolution method is to be provided just to substitute the already proven conventional methods for the target file discovery.

Consequently, the options in **Figure 21** (a) and (b) are encouraged to use for the generic FpUmid applications[13].

It ought to be noted, however, that, behind the FpUmid creation discussed so for, there is an assumption that the source MXF file ("Source.mxf") is the original material assigned with newly created MpUmid and FpUmid, *i.e.*, the UMID with a newly created Mat.# ("XXX" or "YYY") and zero Inst.# values.

On the other hand, there are cases in which the sources MXF file is not the original material but so called derived one, and when this is the case, the option in **Figure 21** (c) is also worthwhile to be chosen in practice.

Figure 22 below schematically demonstrates such a case where a derived MXF file ("Result.mxf"), resulted from the partial retrieval applied to the original MXF file ("Source.mxf") as shown in **Figure 21** (b), is further partially retrieved to create another resulting MXF file ("Result2.mxf").

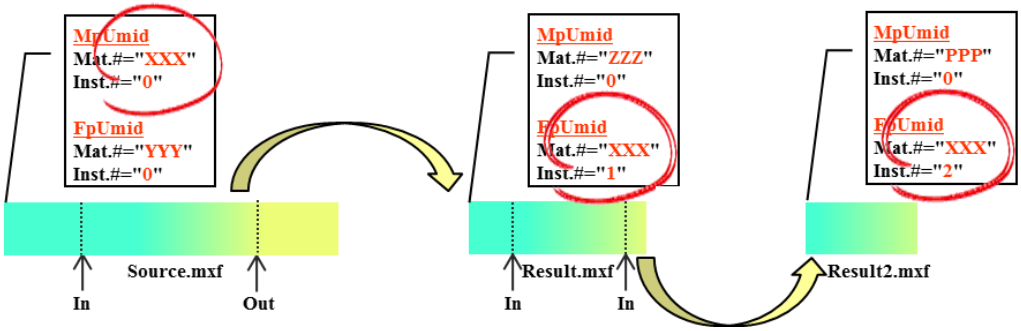


Figure 22: FpUmid application for partial retrieval from a derived MXF file

Since the "Result2.mxf" is also a new MXF file to be managed independently, its MpUmd is newly created with a new Mat.# ("PPP") and zero Inst.# values. Unlike the FpUmid creation method adopted at the first partial retrieval for the "Result.mxf" creation, the FpUmid creation based on the FpUmid of a source MXF file introduced in **Figure 21** (c) is adopted at the second partial retrieval for the "Result2.mxf" creation, where the Mat.# value of FpUmid of the source MXF file ("Result.mxf") is inherited to its Mat.# ("XXX") and another non-zero value than that used for the FpUmid of the source MXF file is newly created for its Inst.# ("2").

[13] If the MpUmid in "Source.mxf" is unreliable as a globally unique material identifier b**ecause** either the origin of "Source.mxf" is unknown, or its UMID value has non-zero Inst.# and/or the UMID "Live stream" flag, the **Figure 21** (a) option as a default behavior is encouraged to be chosen unless the use of such MpUmids is intentionally taken into consideration at the production system design in advance.

While the Inst.# value is significant only whether it is zero (for the original material) or non-zero (for the derived material) in principle, SMPTE ST 330 [1] specifies another interesting Inst.# generation method called “Copy number and 16-bit PRS (Pseudo-Random Sequence) generator”, which provides additional functionality.

When the “Copy number and 16-bit PRS generator” method is adopted, the Inst.# field starts with the copy number field of 1-byte long, which accommodates the number of media processing applied since its original material, immediately followed by a random number field of 2-byte long to distinguish the derived material having the same copy number.

Suppose the “Copy number and 16-bit PRS generator” method is adopted in the FpUmid application for the partial retrieval chain demonstrated in **Figure 22**. The Inst.# then clearly indicates the number of partial retrievals applied since its original MXF file, *i.e.*, “0” for an original MXF file (“Source.mxf”), “1” for the MXF file partially retrieved directly from the original MXF file (“Result.mxf”), and “2” for the MXF file further partially retrieved from the derived MXF file (“Result2.mxf”), as obviously shown in the figure[14]

It is worthwhile to note that, although it is the source MXF file which is referenced back by the FpUmid resolution for the FpUmid creation in **Figure 21** (b), it is the original MXF file for the FpUmid creation in **Figure 21** (c). While they are identical for the derived MXF file directly from the original MXF file, they differ for the derived MXF file derived from another existing derived MXF file, as shown by the “Result.mxf” and “Result2.mxf” in **Figure 22**, respectively.

4.4.3.3 For the MXF file modification at its essence

A similar FpUmid treatment is applicable also to the MXF file modification at its essence. Consider the case where the source MXF file, or “Source2.mxf”, is modified at its essence by overwritten, resulting in a modified MXF file (but its file name remains unchanged) as shown in **Figure 23** below.

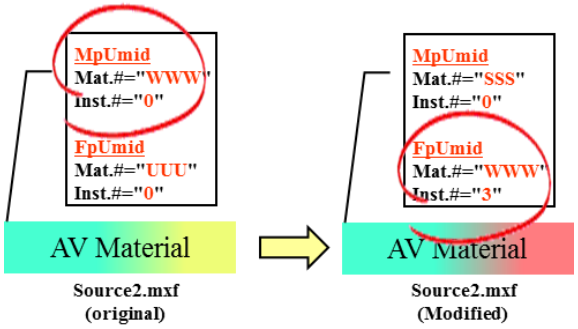


Figure 23: FpUmid application for MXF file modification at its essence

As discussed in Section 4.3.2.2.2, to avoid a collision of the MpUmid of the modified MXF file with that of an MXF file clone to the original one which might exist elsewhere, the MpUmid of the modified MXF file must be a newly created UMID value with its Inst.# being zero. The FpUmid of the modified MXF file, on the other hand, can take the UMID value inherited from the MpUmid of the original MXF file, *i.e.*, its Mat.# is preserved as “WWW” while the Inst.# of the FpUmid being non-zero (“3”) in the case of **Figure 23**.

This kind of FpUmid treatment also brings additional usefulness of the FpUmid similar to that for the partial retrieval discussed in Section 4.4.3.1. Again, as is obvious in the figure, the resolution of the FpUmid of the modified “Source2.mxf” with its Inst.# masked to zero will lead to the URL of the original “Source2.mxf” MXF file before modification if exists elsewhere, though this does not mean that its existence is always guaranteed on the contrary to the case of the partial retrieval.

4.4.3.4 For the file ingest to create an MXF file

The aforementioned FpUmid treatments are applicable even when the source material is an incoming audiovisual stream over, *e.g.*, SDI (Serial Digital Interface), provided the UMID is applied to the stream as has been done in the traditional VTR/SDI-based media production environment (See Section 4.5.2 or Section C.10 of [2]).

[14] A random number that follows the copy number is omitted for simplicity.

In this traditional environment, a bounded sequence of frames is regarded as a material, which is usually created on a tape by the VTR's REC (Start and Stop) operation. Consequently, materials created by multiple REC operations are recorded on a tape in a concatenated fashion.

Because only a part of material such as a frame is accessible at a certain point of time for VTR, every frame in a sequence as a material is recorded together with the UMID on a tape, which uniquely identifies the sequence as a whole. Then, when such a material on the tape is played out, a UMID attached to the frame on a tape is embedded as is into the SMPTE ST 291 Ancillary Data Packet of the frame [19].

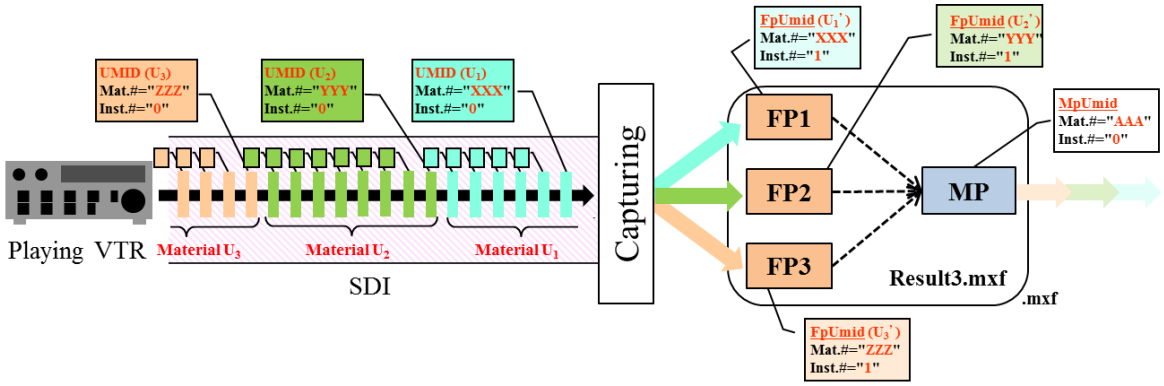


Figure 24: FpUmid application for the file ingest to create an MXF file

The left-hand side of **Figure 24** on the previous page schematically demonstrates the emission of a series of the materials originally recorded on a tape ("Material U_1", "Material U_2" and "Material U_3") with their respective UMIDs (newly created UMIDs of new Material (Mat.#) and zero Instance (Inst.#) Numbers) over the SDI at the VTR playout.

The right-hand side of **Figure 24** then demonstrates how this audiovisual stream over the SDI is captured and recorded in the form of an MXF OP2a file having three File Packages ("Result3.mxf").

Because "Result3.mxf" by itself is a newly created MXF file, the Material Package has the MpUmid of a newly created UMID value with zero Inst.#. Similar to the case of **Figure 20**, on the other hand, each File Package has the FpUmid[15] created based on the UMID of the same value attached to every frame in the incoming bounded sequence of frames as a material to be stored into an Essence Container described by the File Package.

4.4.3.5 For the live feed capture to create an MXF file

When the incoming audiovisual stream is the switched live feed from multiple cameras, the FpUmid application scenario described in Section 4.4.3.4 is modified as follows:

According to SMPTE ST 330 [1], there is a special use of UMID for the live-stream, *i.e.*, for a direct live signal source from a material creation device which has never been recorded, the UMID with "Live stream" flag (the bottom nibble of the byte 12th of UMID being set to "F_h").can be attached to simply signal it. Hence, if such a UMID value is embedded into the stream from a camera (and survives even during the switching operation), the UMID can be used as a hint to uniquely identify a live feed from a particular camera as demonstrated in the left-hand side of **Figure 25**

[15] The apostrophe attached to the original UMID value in the parentheses next to the FpUmid in the figure indicates that this is the UMID of the inherited Mat.# and non-zero Inst.#.

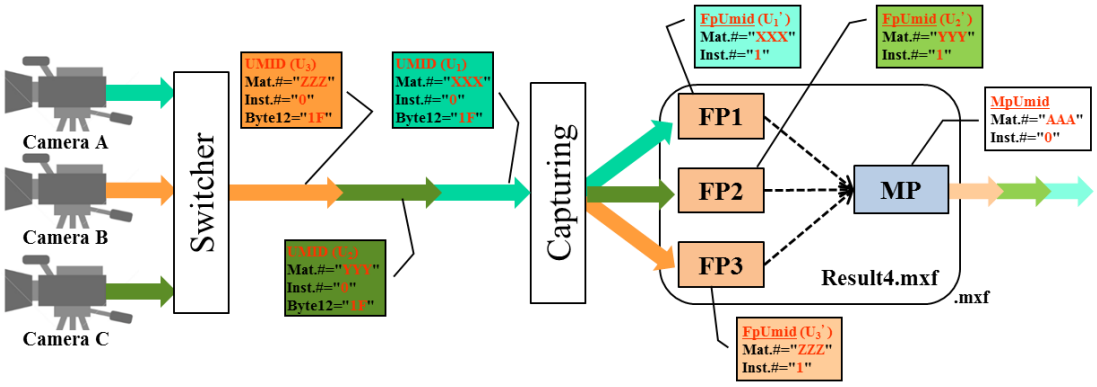


Figure 25: FpUmid application for the live feed capture to create an MXF file

In **Figure 25**, a live feed from "Camera A" is attached with a UMID U_1 in which its Mat.# is set to "XXX" and the bottom nibble of its byte 12th is set to "F_h"[16], and so on. The live feeds supplied from three cameras accordingly are selectively switched to form a single live feed, which is then captured to form an MXF OP2a file having three File Packages ("Result4.mxf").

Because "Result4.mxf" by itself is a newly created MXF file, the Material Package has the MpUmid of a newly created UMID value with zero Inst.#. The File Package, on the other hand, has the FpUmid which is created based on the UMID of the same value attached to every frame in the incoming bounded sequence of frames as a material in the same way as shown in **Figure 24**[17].

Note that the UMID with "Live stream" flag is not the unique material identifier in question but a special UMID use to simply signal the live-stream. As is defined by the UMID Application Principle 1 (Definitions) shown in Section 3.3.2, the material a UMID uniquely identifies must take a persistent and deterministic form so that it can be uniquely identified without ambiguity. This is not applicable to the live-stream because it is neither persistent nor deterministic.

If a constant value is always used for the UMID with "Live stream" flag to uniquely identify a live feed from a particular camera since its initial assignment such as U_1 for the live feed from "Camera A" in **Figure 25**[18], on the other hand, it can provide another benefit than the unique material identifier, *i.e.*, as the unique material "source" identifier.

So what happens when the FpUmid of a resulting MXF file in this case is resolved? Because there is no original material referenced back as a source material in such a way as shown in **Figure 19**, the FpUmid of resulting "Result4.mxf" cannot be resolved in a way by using the standard UMID resolution.

If the UMID with "Live stream" flag for each material "source" is appropriately managed as its unique material "source" identifier, however, the resolution of the FpUmid will lead to the material "source" identification. For example, the FpUmid U_1', attached to the File Package "FP1" in **Figure 25**, will be resolved to a constant U_1, which globally uniquely identifies "Camera A" if the correspondence between the U_1 and "Camera A" is appropriately managed.

It is worthwhile to note that the use of the UMID with "Live stream" flag and its resolution to the material "source" will bring the benefit also to the growing MXF file discussed in Section 4.3.4. If the UMID with "Live stream" flag attached to the incoming source stream is used for the MpUmid of an MXF file under creation, it can not only signal the MXF file growing but also tell which material "source" is used to create this MXF file, though the MpUmid needs to be replaced with a newly created UMID value of zero Inst.# when the MXF file stops growing and becomes persistent.

---

[16] The value "1" at the top nibble of the byte 12th indicates that the "SMPTE method" is used to create the Material Number value according to SMPTE ST 330 [1].

[17] The "F_h" value at the bottom nibble of the byte 12th of UMID must be replaced with a new value indicating the Instance Number generation method according to SMPTE ST 330 [1].

[18] Because most Material Number generation methods require a time stamp at which a UMID containing the Mat.# is newly created according to SMPTE ST 330 Annex A [1], a constant time stamp must be used in order to obtain the constant UMID value regardless of a camera status. This can be achieved by using either a manually preset time stamp value such as the UNIX Epoch time (1970-01-01T00:00:00Z) or a time stamp at which a camera is initially powered on.

Care ought to be taken that there is an assumption in the discussion on the FpUmid applications for the file ingest in Section 4.4.3.4 and the live feed capture in this section that the resulting MXF file to be created can have arbitrary number of File Packages, *i.e.*, it is the MXF OP2a file.

In reality, however, the most widely disseminated MXF file in the industry so far is the simplest one, the MXF OP1a file, which can contain only a single File Package in it. If this is the case, then how to create the FpUmid again depends on the consideration of relationship between the material "source" and resulting MXF files, *i.e.*, while the default behaviour would be to create a new UMID value for the FpUmid[19] as discussed in Section 4.4.2, another might choose the FpUmid creation based on the UMID (with "Live stream" flag) of one of the source materials if it is considered as dominant for the resulting MXF file creation[20].

If the default behavior is applied to the FpUmid creation when the resulting file is an MXF OP1a file, the usefulness of FpUmid is quite limited as is pointed out in Section 4.4.2. However, this shortcoming can be well compensated when the BodyUmid is to be employed appropriately, which is to be discussed in Section 4.5.

4.5 Applications of Body UMID (BodyUmid)

4.5.1 What does the BodyUmid identify?

As is introduced in Section 4.2.2, an MXF file contains either the Basic or the Extended UMID at its File Body, which is called Body UMID (BodyUmid). A typical implementation of the BodyUmid is that it is incorporated into the MXF Generic Container per frame basis, resulting in being associated with each frame. Therefore, it is natural to consider that a BodyUmid uniquely identifies the frame associated with it.

On the other hand, as briefly discussed in Section 4.4.3.4, in the traditional VTR/SDI (Serial Digital Interface) environment where a bounded sequence of frames is regarded as a material, a UMID associated with a frame in the material is used to uniquely identify not only the frame but also the material as a whole. To accommodate those two identifications schemes by a single UMID in the traditional VTR/SDI-based environment, the Extended UMID of 64-byte long is utilized, which is further discussed in Section 4.5.2.2.

4.5.2 Overview of UMID applications in the traditional VTR/SDI-based media production environment

4.5.2.1 Introduction

Before discussing the BodyUmid applications in MXF, it is worthwhile to quickly review what has been done for the UMID in the traditional VTR/SDI-based media production environment.

4.5.2.2 Material identification scheme by the Extended UMID

Figure 26 below schematically illustrates how the Extended UMID composed of the 32-bytes Basic UMID and the 32-bytes Source Pack, which originally describes "When", "Where", and "Who" initially creates a frame, is utilized to uniquely identify a material as a bounded sequence of frames as well as an individual frame in the material.

[19] If no UMID is attached to the incoming audiovisual stream, the default behavior must apply.

[20] To obtain full advantage of the linking back to the source material even in the MXF OP1a file, there is an existing practice for the FpUmid creation in which 1) a UMID value inherited from the UMID attached to the first frame in a stream being captured is temporary assigned to the FpUmid, 2) the value of UMID attached to the subsequent frames are monitored, and 3) if different UMID value from the first one is detected during the monitoring, the FpUmid value is replaced with a newly created UMID value.

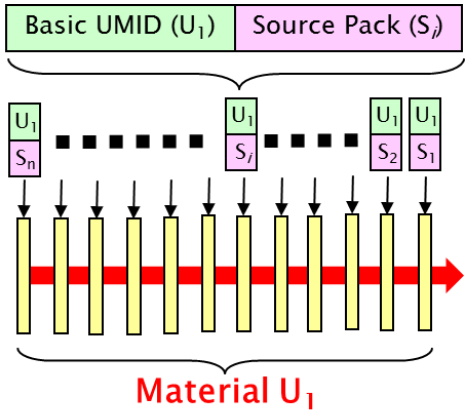


Figure 26: Double layered material identifications by Extended UMID

As shown in the figure, a material is characterized by the Basic UMID, U_1, being attached to all frames in the material, from which the UMID Application Principle 6 (Extended UMID) has derived. Although only a part of material such as a frame is accessible at a certain point of time for the linear recording device such as VTR, this Basic UMID attachment enables for any part of the material to provide the Basic UMID uniquely identifying the material as a whole.

On the other hand, an individual frame in a material is distinguished by the Source Pack, S_i (i = 1-n), immediately following the Basic UMID to form the Extended UMID. Thanks to a finer granularity than the reciprocal of the frame rate for the "When" field, the Source Pack varies at every frame, and therefore, the Extended UMID composed of the Basic UMID uniquely identifying the material as a whole and the Source Pack distinguishing each frame in the material can uniquely identify an individual frame in the material.

Note that while the Extended UMID of 64-byte long is discussed for the material identification scheme in this section, the Basic UMID of 32-byte long only can also be used instead of it, though it just uniquely identifies a material as a bounded sequence of frames as a whole[21].

It ought to be also noted that, in **Figure 26**, while a frame is taken for an example of the Material Unit, a quantum of material uniquely identified by the Extended UMID, any other entities such as a group of frames can be the Material Unit. To be strict, the Material Unit can contain not only a frame but also audio samples that synchronize with the frame at its playout, though a frame is used as a representative of the Material Unit in this discussion.

4.5.2.3 UMID applications at the material creation, playout and dubbing

4.5.2.3.1 Introduction

When multiple materials are created as a result of the VTR's REC (Start and Stop) operations, they are sequentially recorded on a tape, each of which is assigned with a newly created Basic or Extended UMID to be stored next to every frame on the tape. When materials are played out, the Basic or Extended UMID at each frame on a tape is read and embedded as is into the designated position of the output audiovisual stream such as the SMPTE ST 291 Ancillary Data Packet [19] in the case of the SDI.

When a VTR as a recorder receives the incoming audiovisual stream, it detects each frame and a Basic or Extended UMID associated with it, records the frame and the UMID on a tape after the UMID is updated according to one of the ways of UMID applications depicted in **Figure 27** to **Figure 29** shown below.

In the following, by using those figures, three possible UMID applications at the material dubbing are schematically demonstrated. The left-hand side of each figure depicts that three concatenated materials on a tape, each of which is uniquely identified by the UMIDs, U_1, U_2 and U_3, respectively, are played out with their respective UMIDs, which are then recorded on a tape with the updated UMIDs at the right-hand side of the figure. Note that while only the

[21] This is also realized by using the Extended UMID with its Source Pack being zero-filled.

components from the Basic UMID, the Material Number (Mat.#) and the Instance Number (Inst.#), are shown in those figures, all of them are assumed to be the Extended UMIDs whose Source Packs are omitted for simplicity[22].

4.5.2.3.2 For the material dubbing as new materials (Case A)

Figure 27 below shows the Case where the incoming audiovisual materials are dubbed as two new materials, each of which is assigned with a newly created Extended UMID, U_4 and U_5, instead of those attached to the incoming materials. Because they are newly created UMIDs, their Mat.# and Inst.# must be newly created values ("PPP" and "QQQ") and zero, respectively.

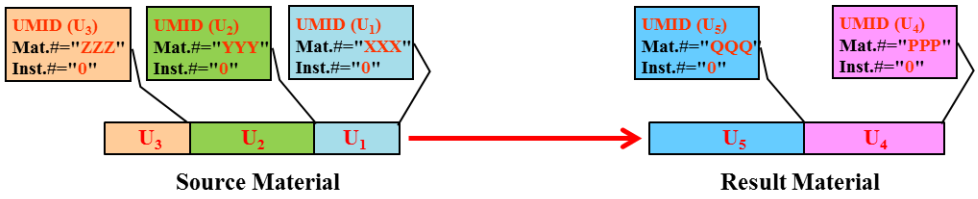


Figure 27: UMID application for the material dubbing as new materials (Case A)

The duration of the resulting materials are left up to the discretion of a device dubbing them, *e.g.*, determined by the REC operations of the recorder VTR. In fact, this is equivalent with a new material creation by the capture of any audiovisual stream including directly from a camera, while the Case in **Figure 27** is regarded as a re-entry of the material to the world of the UMID based material management. It is therefore insignificant at all whether a UMID is attached to the incoming audiovisual stream or not in this Case.

When the Extended UMIDs are attached to the incoming materials, their Source Packs, if exist, are encouraged to inherit as they are even at the creation of the new Extended UMIDs to be stored. Because of a new material creation, however, there is an alternative that Source Packs, which actually reflect the "When", "Where" and "Who" information about the materials being created by this dubbing, are newly created and used for the new Extended UMID creations regardless of the presence of incoming Source Packs. Furthermore, there is also an option to remove the incoming Source Packs for security reasons either by fulfilling zero values to the Source Pack fields for the new Extended UMID creations or by replacing the incoming Extended UMID with a newly created 32-byte Basic UMID only.

4.5.2.3.3 For the material dubbing as inherited materials (Case B)

Figure 28 below shows the Case where the incoming audiovisual materials are dubbed as three inherited materials, each of which is assigned with an inherited Extended UMID, U_1', U_2' and U_3', instead of those attached to the incoming materials. Because they are the inherited UMIDs, their Mat.# and Inst.# must be the same as those in the incoming Extended UMIDs ("XXX", "YYY" and "ZZZ") and certain non-zero values ("3", "2" and "1"), respectively.

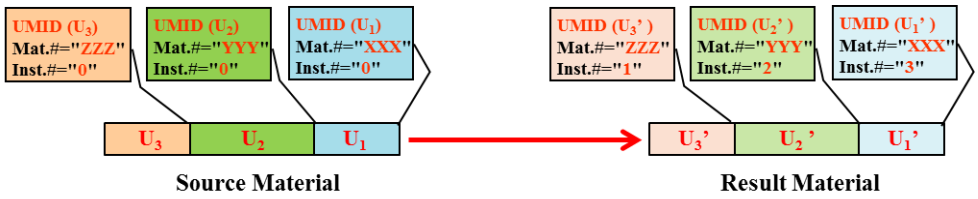


Figure 28: UMID application for the material dubbing as inherited materials (Case B)

The durations of the resulting materials are basically the same as those of the respective materials. Note that it is significant whether UMIDs are attached to the incoming audiovisual stream or not in this Case because this is applicable only when the incoming stream has UMIDs being attached.

When there are UMIDs being attached, this Case is applicable not only to the UMID of a newly created UMID value with the new Mat.# and zero Inst.# as shown in **Figure 28** but also to the UMID of the inherited one with the inherited

[22] They could be also the Extended UMID with zero-filled Source Pack or even the 32-bytes Basic UMID only. In both cases, the discussion made in Section 4.5.2.3 is applicable, though.

Mat.# and the non-zero Inst.#. If the latter is the case, its Inst.# value is to be replaced with another non-zero value while the Mat.# is further inherited as well for the UMIDs to be stored with the resulting material.

Furthermore, this Case is also applicable even to the UMID with "Live stream" flag for the essence stream directly from a camera as a material source, as is discussed in Section 4.4.3.5. Note that because the UMID "Live stream" flag is signaled by the "F_h" value set to the bottom nibble of the byte 12th of UMID which usually specifies the Instance Number (Inst.#) generation method according to ST 330 [1], the resulting material needs to be assigned with an updated UMID in which the UMID "Live stream" flag is replaced with a new value indicating the Inst.# generation method. A non-zero value generated based on the method is then assigned to the Inst.# field of the updated UMID while the Mat.# of the incoming UMID again remains unchanged so that it is referenced back to the material source.

Because of an inherited material creation, the Source Pack in the incoming Extended UMID must be also inherited as is according to the UMID Application Principle 7 (Source Pack), though there still remains an option to remove the incoming Source Pack for security reasons.

It is worthwhile to note that, thanks to an additional function for the copy number management [1], this is the Case which is most frequently used as existing practices in the traditional VTR/SDI-based media production environment because the dubbing via SDI causes quality degradation problem due to the repetition of lossy compression and decompression, while the Case A shown in **Figure 27** is still another option to be chosen for some other reasons.

4.5.2.3.4 For the material dubbing as identical materials (Case C)

Figure 29 below shows the Case where the incoming audiovisual materials are dubbed exactly as they are including their durations as well as their Extended UMIDs being attached.

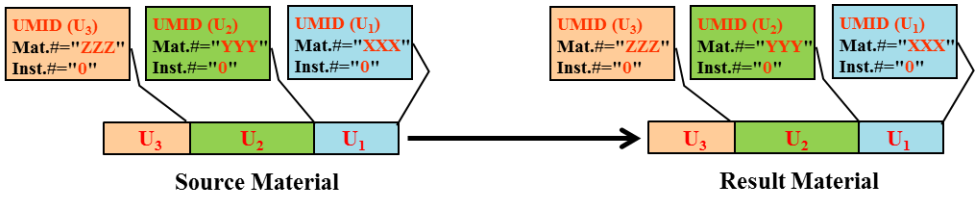


Figure 29: UMID application for the material dubbing as identical materials (Case C)

Because of an identical material creation, the Source Pack at the incoming Extended UMID must be also inherited as is according to the UMID Application Principle 7 (Source Pack), though there still remains an option to remove the incoming Source Pack for security reasons even in this Case.

Note that this Case is applicable only when the resulting material is identical with the incoming one in the sense of the UMID Application Principle 4 (UMID Identification), which is realized by the dubbing via SDTI-CP (Serial Data Transfer Interface – Content Package) [16] in the traditional environment, *i.e.,* this Case is not applicable to the dubbing via conventional SDI that involves the lossy compression and decompression, while the Case A and Case B shown in **Figure 27** and **Figure 28**, respectively, are still options to be chosen for some other reasons even in such a dubbing as that via the SDTI-CP.

4.5.2.3.5 Limitation of UMID applications for the traditional VTR/SDI-based media production environment

Theoretically, the UMID applications in the traditional VTR/SDI-based media production environment discussed in Section 4.5.2.3 is valid. Practically, however, a unique identification of a material as a sequence of frames is hard to be maintained because of a common practice called insert edit operation at VTR.

For the insert edit operation, a part of frames in a material on a tape is overwritten with frames from another material, resulting in bringing two different materials identified by a single UMID. Imagine the material shown in **Figure 26** as the original one, for example. When frames with the Extended UMIDs of U_1S_{i-1} to U_1S_{i+1} in the figure are overwritten with frames from another material, the original material is split into three parts, *i.e.*, the rightmost remaining part composed of frames with Extended UMIDs of U_1S_1 to U_1S_{i-2}, the leftmost remaining part composed of frames with Extended UMIDs of U_1S_{i+2} to U_1S_n, and the overwritten part with UMIDs from another material.

Although the first two parts are completely different materials because of different sequences of frames, they both are resulted in being identified by the same UMID U_1 according to the Extended UMID based material identification discussed in Section 4.5.2.2, which breaches the UMID Application Principle 3 (UMID Integrity)[23].

This is in fact the same situation as that discussed in Section 4.4.3.3, and therefore, the ideal action to be taken is to replace all the UMIDs attached to frames with a newly created UMID value, *e.g.*, U_3S_i. In practice, however, it is unfeasible to be done because it is only the part of frames being overwritten in which the Extended UMIDs can be also replaced with the new UMID values for the VTR. In other words, it is impractical to force the VTR to access all the frames in a material just to replace the Extended UMIDs.

In the existing practices, together with lack of industry standardized tool such as the UMID Resolution Protocol, the UMID is hard to be used as a globally unique material identifier for the material management in the traditional VTR/SDI-based media production environment. On the other hand, another UMID use as a linking tool is still effective. In particular, because of the quality degradation problem caused by the repetition of lossy compression and decompression during editing, the UMID as a linking tool with the copy number management [1] has been only the UMID application appreciated in the traditional VTR/SDI-based media production environment.

4.5.3 Study of BodyUmid applications

4.5.3.1 Introduction

Based on the observation of UMID applications in the traditional VTR/SDI-based media production environment described in Section 4.5.2, this subsection discusses how the BodyUmid in MXF is to be utilized in the modern file-based media production workflow.

4.5.3.2 Logically expected functions of BodyUmid for its applications

As discussed in Section 4.5.2.3.5, there is an obvious limitation for the Extended UMID attached to the frames (as the Material Units) to be used to define a material as a sequence of frames. In the case of MXF, however, the bounded sequence of frames is determined by an Essence Container, which is uniquely identified by the FpUmid in the File Package describing the Essence Container, regardless of the BodyUmids being attached to the frames (as the Edit Units) in the Container[24].

Therefore, it is more practical for the BodyUmid not to be stuck to the double layered material identification scheme depicted in **Figure 26** but to be used as a tool just to characterize an individual Material Unit (as the Edit Unit in an MXF file, which is a frame in this case).

Based on this consideration together with the UMID Application Principles, the logically expected functions of the BodyUmid, as a form of the Extended UMID, are described as follows:

- According to the UMID Application Principle 2 (UMID Creation), if the Inst.# in the basic part of BodyUmid is equal to zero, the Edit Unit attached with it ought to be regarded as an original one or at least as a clone of the original one,
- According to the UMID Application Principle 5 (UMID Inheritance), if the Inst.# in the basic part of BodyUmid is NOT equal to zero (but a certain non-zero value), the Edit Unit attached with it ought to be regarded as the one derived in some way from the original material uniquely identified by the basic part of BodyUmid with its Inst.# masked to zero.

In both cases, when the BodyUmid contains a non-zero Source Pack, the origin of the Edit Unit within the original material can be unambiguously identified. Otherwise, or if the BodyUmid takes a form of the Basic UMID only, what is identified by such a BodyUmid is the original material containing the origin of the Edit Unit somewhere within it.

---

[23] If the Extended UMIDs attached to the frames being overwritten remain unchanged, it seems the UMID U_1 still uniquely identifies the resulting material as a whole. But this still breaches the UMID Application Principle 3 (UMID Integrity) because of difference between the original and the resulting materials at their overwritten part.

[24] While a frame in the Essence Container is called Edit Unit in the MXF viewpoint, it can be also regarded as the Material Unit in the UMID application viewpoint. Although they are usually different aspects of the same entity (a frame in this case), we intentionally call it Edit Unit and Material Unit when it is in the Essence Container of an MXF file and it is on the wire (*e.g.*, via the SDI), respectively.

It is worthwhile to note that the Inst.# based copy number management introduced in Section 4.5.2.3.5 is still applicable even to any BodyUmid application, *i.e.*, the number of media processing applied to an Edit Unit since its origin can be stored as a part of the Inst.# value when an appropriate Inst.# creation method is chosen for every processing step applied to the Edit Unit [1].

Furthermore, while the UMID "Live stream" flag (the bottom nibble of the byte 12th of UMID being set to "F$_h$") discussed in Section 4.4.3.5 is also employed, it can only appear in the Material Unit which has never been recorded so far. In other words, it cannot appear in any BodyUmid because the "Live stream" flag must be reset when a Material Unit with the flag is recorded as the Edit Unit attached with the BodyUmid.

Caution ought to be taken that the original material in the above context is the one which is attached with a newly created UMID of a new Mat.# and zero Inst.# values at its creation regardless of how it is created. In other words, any existing material can be regarded as the original material if its UMID is replaced with a newly created UMID value.

4.5.3.3 BodyUmid applications for a new MXF file creation

4.5.3.3.1 The basic part of BodyUmid treatments

Just like the MpUmid to be newly created at a new MXF file creation from scratch by, *e.g.*, acquisition (See Section 4.3.2.2.2), the BodyUmid needs to be newly created and attached to every Edit Unit (typically, a frame) when the essence is stored into the Essence Container (as a sequence of the Edit Units). Following the conventional treatment of the basic part of Extended UMID discussed in Section 4.5.2.2, it is expected that the basic part of BodyUmid uniquely identifies the original material that contains an Edit Unit with the BodyUmid as a whole. Therefore it is strongly recommended that the value of the basic part of BodyUmid be equal to the MpUmid value[25].

If an incoming Material Unit already has its own Extended UMID being attached in such a way as shown in **Figure 24**, following the traditional Extended UMID applications discussed in Section 4.5.2.3, there are three options to be taken for the (basic part of) BodyUmid, *i.e.*,

a) To replace the incoming UMID with a newly created BodyUmid,

b) To inherit the incoming UMID by inheriting its Mat.# and replacing its Inst.# with another non-zero value for the BodyUmid,

and

c) To reuse the incoming UMID as is as the BodyUmid,

depending on how the Material Unit is to be stored as the Edit Unit in the Essence Container.

Specifically, if an incoming Material Unit such as a frame is the baseband signal via SDI and it is lossy compressed when stored into the Essence Container as the Edit Unit, either the options a) or b) can be chosen because the resulting Edit Unit stored in the Essence Container cannot be identical with the incoming Material Unit in the sense of the UMID Application Principle 4 (UMID Identification). If the incoming Material Unit is already the compressed data such as via SDTI-CP or IP and it is stored exactly as is into the Essence Container as the Edit Unit, any of the options a) to c) can be chosen.

In addition,

d) To delete the incoming UMID,

is still an option to be chosen if an incoming Material Unit is desired to be stored into the Essence Container as the Edit Unit without the BodyUmid for security reasons.

Table 1 below summarizes the (basic part of) BodyUmid treatments that can be chosen depending on how to store the Material Unit.

[25] To be strict, because the MpUmid is the Basic UMID of 32 byte long while the BodyUmid in question is the Extended UMID of 64 byte long, their Length fields (the 13th byte of UMID) take different values, *i.e.*, 13$_h$ and 33$_h$ for the MpUmid and the BodyUmid, respectively.

Table 1: BodyUmid treatments

Does incoming UMID exist?	How to store an incoming Material Unit as the Edit Unit?	BodyUmid treatment	Resulting BodyUmid value
No	Lossy compressed (*e.g.*, via SDI)	Newly created	New
		Nothing done	-
	As is (*e.g.*, via SDTI-CP or IP)	Newly created	New
		Nothing done	-
Yes	Lossy compressed (*e.g.*, via SDI)	Replaced	New
		Inherited	Inherited
		Deleted	-
	As is (*e.g.*, via SDTI-CP or IP)	Replaced	New
		Inherited	Inherited
		Reused	As is
		Deleted	-
Yes (w/ "Live stream" flag)	Lossy compressed (*e.g.*, via SDI)	Replaced	New
		Inherited (w/o "Live stream" flag)	Inherited
		Deleted	-
	As is (*e.g.*, via SDTI-CP or IP)	Replaced	New
		Inherited (w/o "Live stream" flag)	Inherited
		Deleted	-

where each term in the "BodyUmid treatment" column denotes as follows:

- "**Newly created**" : A new BodyUmid is created with a new Mat.# and zero Inst.# values and stored with an incoming Material Unit as the Edit Unit,
- "**Replaced**" : A new BodyUmid is created with a new Mat.# and zero Inst.# values and stored with an incoming Material Unit as the Edit Unit instead of the UMID originally attached to the Material Unit,
- "**Inherited**" : A BodyUmid is created by inheriting the Mat.# of the UMID attached to an incoming Material Unit and by replacing the Inst.# with another non-zero value, and stored with the Material Unit as the Edit Unit,
- "**Reused**" : A UMID attached to an incoming Material Unit is stored as is with the Material Unit as the Edit Unit,

- "**Deleted**": A UMID attached to an incoming Material Unit is deleted and only the Material Unit is stored as the Edit Unit.

As already mentioned above, when the BodyUmid is newly created, its basic part is equivalent with the MpUmid which is newly created in order to globally uniquely identify the MXF file being created. In other words, the newly created Mat.# value must be shared among the MpUmid and the BodyUmid[26].

Furthermore, when the FpUmid is created based on the UMID attached to every incoming Material Unit as demonstrated in **Figure 24**, the basic part of BodyUmid to be created by inheriting the UMID attached to an incoming Material Unit is expected to be equivalent with the FpUmid[27]. In other words, the newly created non-zero Inst.# value ought to be shared among the FpUmid and the BodyUmid[28]..

It needs to be also mentioned that if the UMID attached to an incoming Material Unit has the UMID "Live stream" flag (the bottom nibble of the byte 12th of UMID being set to "F_h") as a result of its direct transmission from a material source such as a camera, the flag must be replaced with a new value indicating a certain Inst.# generation method [1] even when the BodyUmid to be stored is created by inheriting the Mat.#. This is because the UMID with "Live stream" flag is used just as a tool to signal that a Material Unit attached with it has never been recorded so far, and therefore, it needs to be removed for the BodyUmid when the Material Unit is stored into the Essence Container as the Edit Unit with it.

4.5.3.3.2 The Source Pack in the BodyUmid treatments

The Source Pack in the BodyUmid is treated in the same way as that for the traditional Extended UMID applications discussed in Section 4.5.2.3. According to the UMID Application Principle 7 (Source Pack), the Source Pack treatment basics are itemized as follows:

- The Source Pack can exist only as a part of the BodyUmid (as a form of the Extended UMID),
- In principle, the Source Pack is reused as is regardless of the basic part of BodyUmid treatment,
- When the basic part of BodyUmid is newly created, its Source Pack can be also newly created if desired,
- The Source Pack can be deleted regardless of the basic part of the BodyUmid treatment if desired.

Based on those Source Pack treatment basics, the Source Pack in BodyUmid treatments that can be chosen depending on the basic part of BodyUmid treatment are summarized in **Table 2** below:

Table 2: Source Pack treatments

Does incoming UMID exist?	Does incoming UMID have Source Pack?	The basic part of BodyUmid treatment	Source Pack treatment	Note
No	No	Newly created	Newly created	New Extended UMID to be stored as the BodyUmid

[26] If the MpUmid is used to signal an MXF file growing as discussed in Section 4.3.4, its Mat.# value is different from that in the BodyUmid during growing. In this case, the newly created Mat.# for the BodyUmid ought to be reused for the MpUmid to be created at the completion of the MXF file creation.

[27] This is effective only when one can produce the File Packages of as many as the number of incoming bounded sequences of Material Units, or when it is the MXF OP2a file (See **Figure 24**). If more than one bounded sequence of Material Units is incoming for the creation of MXF OP1a file that can contain only a single File Package, the FpUmid needs to be newly created with a new Mat.# and zero Inst.# values, which differs any BodyUmid that inherits the UMID attached to incoming Material Units.

[28] Theoretically, if an incoming bounded sequence of Material Units is stored completely as is (in the sense of the UMID Application Principle 4 (UMID Identification)) into a single Essence Container, the UMID attached to incoming Material Units with zero Inst.# can be reused as is for the value of FpUmid uniquely identifying the Essence Container. Practically, the UMID value with a new non-zero Inst.# for the FpUmid is more feasible to be used even in this case, though.

			Nothing done	New Basic UMID to be stored as the BodyUmid
		Nothing done	Nothing done	No BodyUmid to be stored
Yes	No	Replaced	Newly created	New Extended UMID to be stored as the BodyUmid
			Nothing done	New Basic UMID to be stored as the BodyUmid
		Inherited	Nothing done	New Source Pack cannot be according to UMID Application Principle 7 (Source Pack)
		Reused		
		Deleted		
	Yes	Replaced	Newly created	New Extended UMID to be stored as the BodyUmid
			Reused	New Basic UMID w/ incoming Source Pack to be stored as the BodyUmid
			Deleted	New Basic UMID to be stored as the BodyUmid
		Inherited	Reused	Inherited Basic UMID w/ incoming Source Pack to be stored as the BodyUmid
			Deleted	Inherited Basic UMID to be stored as the BodyUmid
		Reused	Reused	incoming Extended UMID to be stored as the BodyUmid
			Deleted	incoming Basic UMID to be stored as the BodyUmid
		Deleted	Deleted	No BodyUmid to be stored

where each term in the “Source Pack treatment” column denotes as follows:

- “**Newly created**” : A new Source Pack is created and appended to the basic part of BodyUmid,
- “**Reused**” : A Source Pack in the incoming Extended UMID is reused as is and appended to the basic part of BodyUmid,
- “**Deleted**”: A Source Pack in the incoming Extended UMID is deleted.

It ought to be noted that the original intention of the Source Pack introduction is that it is expected to be used to track the Material Unit that travels through the production workflow chain so that the audit trail can be maintained.

Therefore the Source Pack is strongly encouraged to be kept as is since its initial creation regardless of the treatment of the basic part of BodyUmid.

4.5.3.4 BodyUmid applications for an existing MXF file modification at its essence

4.5.3.4.1 BodyUmid application for the Edit Unit modification

Suppose there is an original source MXF file that contains the Essence Container composed of five Edit Units, each of which is uniquely identified by the BodyUmid (as the Extended UMID) from U_1S_1 to U_1S_5, respectively. Because this is an original material, the MpUmid globally uniquely identifying the MXF file as a whole is also U_1 according to the discussion made in Section 4.5.3.3.1.

When a part of the Essence Container, say, composed of two Edit Units originally identified by U_1S_3 and U_1S_4, is modified on their own by, *e.g.*, superimposing text or the color grading, the MpUmid must be replaced with a new UMID value (*e.g.*, U_2) in order to obey the UMID Application Principle 3 (UMID Integrity). Furthermore, the BodyUmid attached to the original Edit Unit must be also updated to an appropriate UMID value. Note that this BodyUmid update is also based on the BodyUmid applications discussed in Section 4.5.3.3 except that the original BodyUmid (as an Extended UMID) before modification is used instead of the Extended UMID attached to an incoming Material Unit (See the "Does incoming UMID exist?" column[29] in **Table 1**).

Figure 30 below shows the case where the MpUmid and FpUmid are updated to "U_2" and "U'_1", respectively, following the discussion in Section 4.4.3.3. The basic part of BodyUmid is then updated by the "Inherited" (from the original BodyUmid) while its Source Pack is "Reused" according to **Table 1** and **Table 2**, respectively. Note that while this is the most preferred treatment of the BodyUmid, other options shown in those Tables (except for the "Reused" in **Table 1**) can be chosen[30] as appropriate for some reasons.

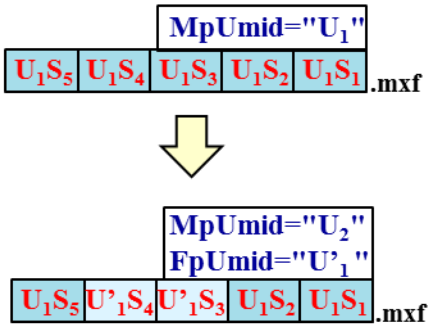


Figure 30: BodyUmid application for the Edit Unit modification

4.5.3.4.2 BodyUmid application for the Edit Unit replacement

Figure 31 on the next page shows the case where a part of the Essence Container is modified by replacing the original Edit Units with those coming from another material globally uniquely identified by the MpUmid U_3. Specifically, in addition to the conventional MpUmid and FpUmid updates, the original Edit Units with the BodyUmid of U_1S_3 and U_1S_4 are replaced with those with U'_3S_6 and U'_3S_7, respectively.

In this case, the BodyUmid treatments discussed in Section 4.5.3.3 are applicable as they are because the resulting Edit Units at the modified part of the Essence Container are regarded as a "new material creation", where the term "material" means the Edit Units[31].

While **Figure 31** shows the most typical case where the basic part of Extended UMID attached to an incoming Material Unit from another material is "Inherited" because of, *e.g.*, lossy compressed recording from the SDI, and its Source Pack is "Reused" as is according to **Table 1** and **Table 2**, respectively, other options shown in those Tables

[29] Because the original Edit Unit in this example is the one already recorded, the UMID with "Live stream" flag is not applicable.

[30] If the "Newly created" or "Replaced" in **Table 1** is chosen, the basic part of BodyUmid must be also U_2.

[31] The Source Pack attached to the original Edit Unit to be replaced is no more effective because of the absence of Edit Unit itself.

can be chosen as needed including to assign a newly created UMID value that is equivalent with the MpUmid (*e.g.*, U_2) to the basic part of BodyUmid instead of inheriting the incoming one (U_3) for some reasons.

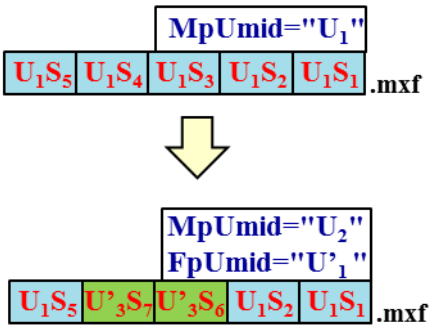


Figure 31: BodyUmid application for the Edit Unit replacement

4.5.3.4.3 BodyUmid application for the Edit Unit insertion

In the case of the modern file-based media production workflow, such an Edit Unit treatment as to insert new Edit Unis into already existing ones in the Essence Container is possible, which has been infeasible in practice in the traditional VTR/SDI-based media production environment.

Figure 32 below shows the case where two Edit Units coming from another material uniquely identified by the MpUmid U_3 are inserted between the original Edit Units uniquely identified by U_1S_4 and U_1S_5.

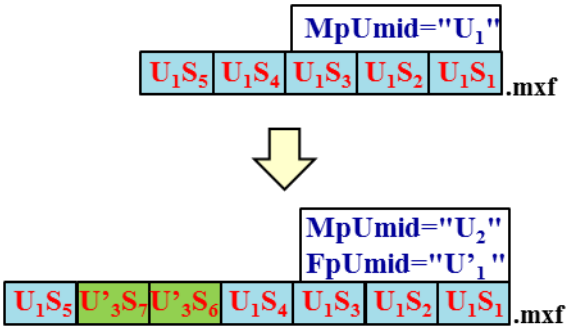


Figure 32: BodyUmid application for the Edit Unit insertion

Similar to the discussion in Section 4.5.3.4.2, since the resulting Edit Units inserted between the existing ones in the Essence Container are also regarded as a “new material creation”, the BodyUmid treatments discussed in Section 4.5.3.3 are applicable as they are.

4.5.3.4.4 BodyUmid application for the Edit Unit processing for the A/B-roll

Figure 33 on the next page shows the case where another material composed of the Edit Units with U_3S_6 to U_3S_9 is spliced to the original source MXF file with the two Edit Units being overlapped. When this is the case, where the Edit Unit is not overlapped, the original Edit Unit with its BodyUmid is kept as is even for another material being spliced under the assumption that the Edit Unit from another material is appended exactly as is. Because this is also based on the BodyUmid treatments discussed in Section 4.5.3.3, while this is the most preferred BodyUmid treatment, any other options from **Table 1** can be chosen as appropriate. For example, if the Material Units from another material is supplied via the SDI and appended to the original source as the Edit Units in a lossy compressed fashion, the BodyUmid is updated by the “Inherited”, *i.e.*, as U'_3S_8 to U'_3S_9 in this case.

The assignment of BodyUmid to the resulting Edit Unit composed of multiple source Edit Units, or to the part where the source Edit Units are overwrapped in **Figure 33**, is somewhat controversial. When one of the source Edit Units is chosen as the main Edit Unit for the composition, however, the BodyUmid treatments discussed in Section 4.5.3.3 is applicable in the same way as discussed in Section 4.5.3.4.1.

In the case of **Figure 33**, the main Edit Unit is chosen according to the degree of contribution of each Edit Unit to compose a resulting Edit Unit. Consequently, the source Edit Unit with U_1S_4 and that with U_3S_7 are chosen as the main Edit Unit for the fourth and fifth resulting Edit Units, respectively. The BodyUmid to be attached to them are created by updating those original BodyUmid to U'_1S_4 and U'_3S_7 by the "inherited" treatment, which are then assigned to respective resulting Edit Units.

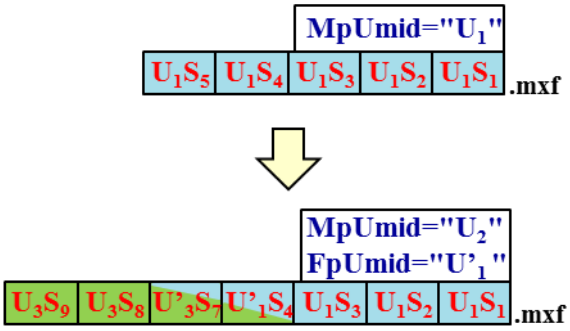


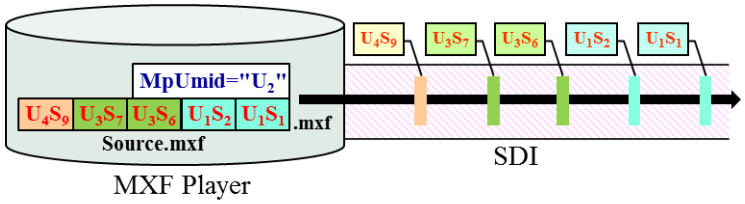
Figure 33: BodyUmid application for the Edit Unit processing for the A/B-roll

It the resulting Edit Unit is a creating one, reliant on the spawning of ideas, conceptual thought and human creativity, it might be reasonable to assign a newly created BodyUmid to it (U_2 in this case to align the MpUmid value). It ought to be noted that this is still an option to be chosen from **Table 1** that summarizes the BodyUmid treatments.

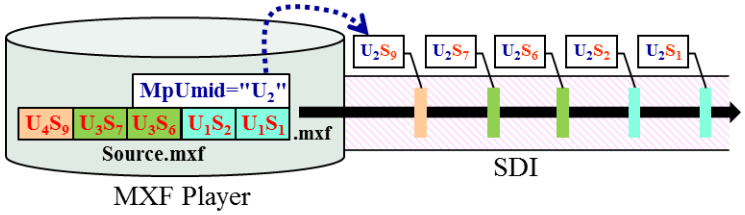
4.5.3.5 BodyUmid applications for an existing MXF file playout

Following the UMID treatment at the material playout in the traditional VTR/SDI-based environment discussed in Section 4.5.2.3, when an existing MXF file is played out, the BodyUmid stored for each Edit Unit (*e.g.*, frame) in the MXF file is read and embedded as is into the designated position of the output audiovisual stream. **Figure 34** (a) below schematically demonstrates how the BodyUmid stored in an MXF file called "Source.mxf" is treated when the MXF file is played out for the SDI output.

Note that this is a default behavior which well harmonizes with the UMID treatment even in the traditional VTR/SDI-based environment, In other words, when multiple VTRs are connected via the SDIs in the traditional VTR/SDI-based media production system, an MXF recorder/player that takes this default behavior can substitute for any VTR in the system. This is therefore suitable for the migration period from the traditional system to the modern file-based one.



(a) To embed BodyUmid as is (by default)



(b) To use the MpUmid value for the basic part of BodyUmid

Figure 34: BodyUmid application for an exisitng MXF file playout

In addition to the default behavior, there is another option in which the Extended UMID to be embedded into the output audiovisual stream is created by replacing the basic part of BodyUmid with the MpUmid value. **Figure 34** (b) above schematically demonstrates this behavior in which the MpUmid value, U_2, is used for the basic part of Extended UMID in the stream regardless of the original BodyUmid value in the MXF file.

This behavior is regarded as a step that temporarily treats an MXF file as if it is an original material newly created from scratch. In fact, if the MXF file is truly created from scratch, in which both the MpUmid and the basic part of BodyUmid share a single value (the UMID Material Number) as discussed in Section 4.5.3.3.1, this behavior is completely equivalent with the default behavior shown in **Figure 34** (a).

Furthermore, it is also worthwhile to note that this behavior is applicable even to an MXF file that has no BodyUmid at all, though only the Basic UMID is embedded into the output audiovisual stream because of the MpUmid that allows the Basic UMID type value only[32].

So far, although an MXF player is assumed to always treat the BodyUmid in an MXF file, it is in fact not mandatory. Specifically, it is the discretion of the MXF player whether to entirely use the BodyUmid as an Extended UMID, to use only the basic part of BodyUmid as a Basic UMID, or not to use at all in order to suppress embedding the BodyUmid for security reasons.

4.5.4 Some considerations on the MXF Operational Pattern (OP) dependent BodyUmid treatments

4.5.4.1 BodyUmid treatments for the MXF OP1a file

4.5.4.1.1 For MXF OP1a AV interleaved, frame-wrapped file

As for the MXF OP1a AV interleaved, frame-wrapped file, which is the most widely available one in the industry at the time of this writing, how to store the BodyUmid in the Essence Container of the MXF file has been already industry standardized by adopting a tool called Generic Container [14, 15] as is introduced in Section 4.2.2.

In this case, the BodyUmid is assigned to every Edit Unit which contains not only the picture essence as a frame but also the sound essence(s) as well as (non-AV) data essence(s) which synchronize with the frame. As a result, when the BodyUmid uniquely identifying an Edit Unit is newly created[33], *i.e.*, with a newly created Mat.# and zero Inst.# values, the UMID Material Type (called MatType, hereafter) in the BodyUmid, or the byte 11^{th} of the UMID UL, must be "$0D_h$", indicating that the Edit Unit is a "mixed group of components in a single container", according to SMPTE ST 330 [1] (which is denoted "AV", hereafter).

When the BodyUmid is created by inheriting the Extended UMID attached to an incoming Material Unit to be store as the Edit Unit, the Inst.# value in the Extended UMID is replaced with another non-zero value to form the BodyUmid. While the MatType in the incoming Extended UMID is usually expected to be "AV", there might be a case where the MatType is not "AV". This comes from the design decision taken in order to realize the logically expected functions of BodyUmid proposed in Section 4.5.3.2. See Section 4.5.4.1.2 (next section) for more discussion on this issue.

In either case, the BodyUmid assigned to an Edit Unit is stored in an MXF file according to the standard way introduced in Section 4.2.2, and it is then transferred as is as a default behavior when the Edit Unit is played out such as to the SDI as discussed in Section 4.5.3.5.

Note also that, as for the Source Pack treatments at the BodyUmid creation, those discussed in **Table 2** of Section 4.5.3.3.2 are applicable as they are.

4.5.4.1.2 For MXF OP1a single essence file

In addition, it ought to be noted that the Generic Container introduced in Section 4.2.2 is also applicable even to the BodyUmid for an MXF OP1a single essence file such as the picture only, frame-wrapped one, when only the System and Video Items shown in **Figure 17** are taken into consideration.

If this is the case, the BodyUmid is assigned to every Edit Unit which contains only the picture essence as a frame. Therefore, when the BodyUmid uniquely identifying the Edit Unit is newly created from scratch as shown in the left-

[32] Because of the definition of Source Pack, its temporary creation for a specific playout cannot be permitted.

[33] According to Section 4.5.3.3.1, the newly created BodyUmid must be equivalent with the MpUmid which globally uniquely identifies the MXF file as a whole.

half of **Figure 35** below, its MatType in the UMID UL needs to specify "06_h", indicating that the target material has "two or more picture components in a single container"[34] according to SMPTE ST 330 [1] (which is denoted "V", hereafter),.

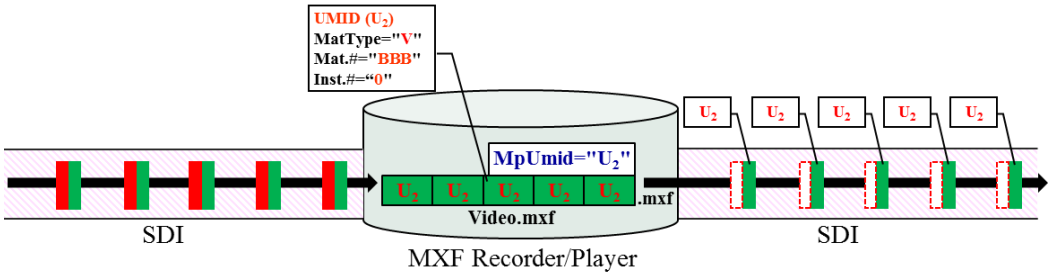


Figure 35: BodyUmid treatments for MXF OP1a single essence file (no incoming Extended UMID)

It is worthwhile to note that when the BodyUmid is newly created, its value must be equivalent with the MpUmid value (U_2 in **Figure 35**) which globally uniquely identifies the MXF file as a whole ("Video.mxf" in **Figure 35**) according to Section 4.5.3.3.1. Consequently, an external application that desires to treat the MXF file can obtain the type of material contained in the file in advance from the MatType in the MpUmid, which is used for the UMID based material management discussed in Section 4.3.2 and thus made accessible from the application as a globally unique material identifier of the MXF file.

When the Edit Unit is played out to the SDI, the BodyUmid stored with it is also transferred as is to the SDI regardless of how to create the outgoing Material Unit on the wire as shown in the right-half of **Figure 35**. It might seem strange to see the UMID with the "V" MatType traveling through the SDI which is supposed to deliver the synchronized "AV" baseband signal. But this ought to be understood that the UMID (with its zero Inst.#) on the wire uniquely identifies only the picture component of the Material Unit resulted from a playout of the corresponding Edit Unit[35], and it also tells that the picture component is identical with that stored in the Edit Unit of the original MXF file ("Video.mxf" in **Figure 35**) in the sense of the UMID Application Principle 4 (UMID Identification), which is a logical consequence of the UMID interpretation based on the logically expected functions of BodyUmid proposed in Section 4.5.3.2.

So what happens to the BodyUmid at the recorder side when it is created from such a UMID attached to an incoming Material Unit on the wire? According to the BodyUmid treatments shown in **Table 1** of Section 4.5.3.3.1, there are three options to be taken for the BodyUmid creation when the Material Unit with the UMID is recorded as the Edit Unit with the BodyUmid in an MXF file.

When the BodyUmid is newly created for the "Newly created" or "Replaced" in **Table 1**, the MatType in the BodyUmid ought to take an appropriate value that correctly reflects the material type of the Edit Unit being created.

When the BodyUmid is created by the "Inherited" from the UMID attached to a Material Unit because, for example, it is recorded as the Edit Unit in the lossy compressed fashion, the BodyUmid inherits the Mat.# and the "V" MatType from the UMID while its Inst.# being set to non-zero. In this case, the BodyUmid (with its non-zero Inst.#) does not uniquely identify the Edit Unit but just tells that the picture component of the Edit Unit has been derived from the original material uniquely identified by the BodyUmid with its Inst.# masked to zero[36]. In other words, the MatType in

[34] The picture component in this context means such as Y component for the YCbCr color subsampling. As a result, "06_h" is specified for normal color picture essence while "05_h" is for the gray scale (Y component only).

[35] There might exist an argument that the UMID to be attached to the outgoing Material Unit to the SDI ought to always have the "AV" MatType rather than the "V" to harmonize with the SDI characteristics. Although it requires additional UMID treatment of changing the MatType value from "V" to "AV" at its transfer, it does not pay at all, *i.e.*, even when the BodyUmid is created by inheriting such a UMID attached to an incoming Material Unit at the recorder side, it cannot be a linking tool to the original material as is because of different MatType value in the BodyUmid ("AV") from that in the MpUmid of the original material ("V"). Note that this cannot be addressed by an application desiring the original material because there is no standard way for the application to obtain the original material type in advance except for inspecting the MatType in the MpUmid.

[36] Theoretically, a problem of unclear correspondence between multiple components of the same material type and their respective BodyUmids occurs when all of them are stored in a single Edit Unit such as those for the sound essence, which ought to

the BodyUmid with non-zero Inst.# value does not indicate the material type of the Edit Unit with the BodyUmid but that of the original material used to form the Edit Unit through some media processing.

When the BodyUmid reuses the UMID attached to the Material Unit by the “Reused” in **Table 1** because it is recorded as the Edit Unit exactly as is, the BodyUmid is also equal to the UMID. In this case, the BodyUmid (with its zero Inst.#) again uniquely identifies only the picture component of the Edit Unit resulted from a recording of the Material Unit, and it also tells that the picture component is identical with that of the Material Unit, and thus even identical with that of the corresponding Edit Unit of the original MXF file (“Video.mxf” in **Figure 35**), in the sense of the UMID Application Principle 4 (UMID Identification), based on the logically expected functions of BodyUmid discussed in Section 4.5.3.2.

On the other hand, when the BodyUmid is created by the “Inherited” from the Extended UMID of the “AV” MatType attached to an incoming Material Unit on the wire as shown in the left-half of **Figure 36** below, the BodyUmid treatment discussed in Section 4.5.4.1.1 is still applicable as is, *i.e.*, the BodyUmid to be assigned to the Edit Unit is created by just replacing the Inst.# value in the Extended UMID with another non-zero value, resulting in inheriting the Mat.# and the “AV” MatType from the Extended UMID.

It seems again strange to see that the BodyUmid of the “AV” MatType is assigned to the Edit Unit of the picture component only. But this ought to be understood that the BodyUmid (with its non-zero Inst.#) does not uniquely identify the Edit Unit but just tells that the Edit Unit is related to the original material uniquely identified by the BodyUmid with its Inst.# masked to zero in a certain way, such as for the Edit Unit to contain a picture component derived from the original material, according to the logically expected functions of BodyUmid in Section 4.5.3.2. In other words, the MatType in the BodyUmid with non-zero Inst.# value again does not indicate the material type of the Edit Unit with the BodyUmid but that of the original material used to form the Edit Unit[37] in a certain way.

As for the Source Pack treatments in the incoming Extended UMID, on the other hand, those discussed in **Table 2** of Section 4.5.3.3.2 are applicable as they are in any case. Hence, both **Figure 35** and **Figure 36** schematically demonstrate the most typical Source Pack treatment specified by the “Reused” in the table, resulting in that the incoming Source Packs, S_j (j=1..5), are preserved as they are for all the BodyUmids to be assigned to respective Edit Units in the figures.

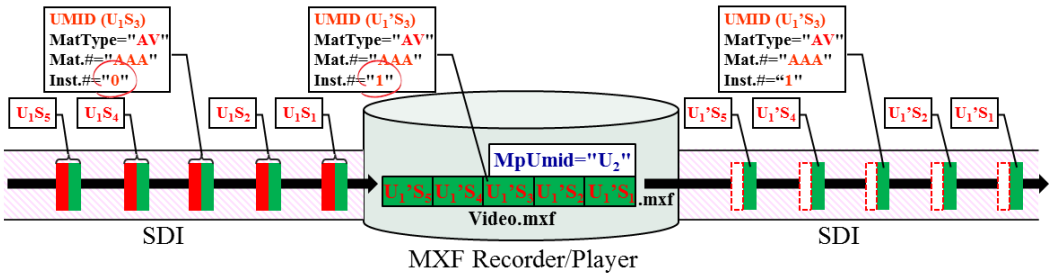


Figure 36: BodyUmid treatments for MXF OP1a single essence file (w/ incoming Extended UMID)

When the Edit Unit is played out to the SDI, the BodyUmid stored with it is also transferred as is to the SDI regardless of how to create the outgoing Material Unit on the wire as shown in the right-half of **Figure 36**.

4.5.4.2 UMID treatments for media processing of Material Unit

When **Figure 36** is observed from the viewpoint of the Material Unit treatments, it is interesting to see that the MXF Recorder/Player at the center of the figure can work as a sort of media filter that produces an outgoing Material Unit with the updated Extended UMID by applying certain media processing to an incoming Material Unit. More specifically, the cylinder box in **Figure 36** behaves as a filter that extracts the picture component of the incoming Material Unit and

---

be addressed by the container specification for the Edit Unit. In reality, however, because only a single UMID per Material Unit is permitted to transfer via the SDI, it can be devoted only either to “AV” or “V”, resulting in making the correspondence obvious.

[37] Because of different material types between the incoming Material Unit (“AV”) and the Edit Unit (“V”) being created, there is no option for the UMID attached to the Material Unit to be “Reused” for the BodyUmid. Therefore, the BodyUmid creation by the “Inherited” in **Table 1** is the only choice if the incoming UMID is desired to be used for the BodyUmid creation.

lossy compresses and decompresses it for the outgoing Material Unit. It also filters out the sound component of the incoming Material Unit and leaves a footprint of the media processing by replacing the Inst.# of incoming Extended UMID with a new non-zero value for the Extended UMID to be attached to the outgoing Material Unit.

Regarding the creation of a new Inst.# value, some Inst.# generation methods have been already specified in Table 4 of SMPTE ST 330 [1], among which the “Local registration” method allows a user to assign arbitrary Inst.# value that meets his/her specific requirements per its own registration basis. Furthermore, similar to the copy number management of the Edit Unit discussed in Section 4.5.3.2, when the “Copy number and 16-bit PRS (Pseudo-Random Sequence) generator” method in the table is chosen and consistently applied over the media processing workflow chain, the number of media processing applied to a Material Unit since its origin can be stored as a part of the Inst.# value, and therefore, the generation comparison among the Material Units sharing the same MatType and the Mat.# in the UMID is realized.

As for the Source Pack treatments in the incoming Extended UMID, on the other hand, those discussed in **Table 2** of Section 4.5.3.3.2 are applicable as they are. Hence, **Figure 36** schematically demonstrates the most typical Source Pack treatment specified by the “Reused” in the table, resulting in that the incoming Source Packs, S_j (j=1..5), are preserved as they are for all the Extended UMIDs attached to respective outgoing Material Units in the figure.

4.5.4.3 BodyUmid treatments for the MXF OP Atom files

As introduced in Section 2.4.6, the MXF OP Atom files are a set of MXF files, each of which contains only a mono type of essence, resulting in video and audio essence data being separately stored in different MXF files. While no constraint is specified to the Material Package for the MXF OP Atom files [12], the most typical use of the Material Packages in the files is that a single Material Package referencing not only to the File Package contained in the same file but also to those contained in other external files is shared by all the MXF OP Atom files as shown in **Figure 10**.

At the time of this writing, there is no standard method on how to store the BodyUmid in the MXF OP Atom files. When the audiovisual essence synchronization is taken into consideration, it would be ideally the Material Package which ought to store a series of BodyUmids, each of which is assigned to an Edit Unit that is supposed to contain the synchronized picture and sound components. However, because all MXF OP Atom files have a duplicate Material Package as mentioned above, it would be too cumbersome to be managed if the Material Package is further requested to contain a bunch of BodyUmids within it.

Taking account of a single essence to be stored in an MXF file also for the MXF OP Atom files, on the other hand, it would be reasonable to consider the adoption of the Generic Container applied to the MXF OP1a single essence file discussed in Section 4.5.4.1.2. In the following, assuming the BodyUmid to be stored in one of the MXF OP Atom files composed of the picture only, frame-wrapped Edit Units via the Generic Container, the BodyUmid treatments are discussed based on the creation of a set of MXF OP Atom files from a series of incoming Material Units shown in **Figure 35** and **Figure 36**.

When a set of MXF OP Atom files are newly created from scratch by, *e.g.*, acquisition (See Section 4.3.2.2.2), a single MpUmid is newly created and shared via the Material Packages in all the MXF OP Atom files. Note that because this is another representation of a synchronized audiovisual material in MXF, the MpUmid shared by the MXF OP Atom files globally uniquely identifies the material composed of a set of those MXF OP Atom files as a whole.

The BodyUmid created in one of the ways shown in **Table 1** of Section 4.5.3.3.1 is then to be stored with the Edit Unit which contains only the picture essence as a frame in the Generic Container of the MXF OP Atom file.

When the BodyUmid is created by the “Inherited” from the Extended UMID attached to an incoming Material Unit to be recorded as the Edit Unit in the MXF OP Atom file similar to those shown in the left-half of **Figure 36**, the BodyUmid is treated in a similar way as discussed in Section 4.5.4.1.2, including the BodyUmid transfer at the playout of the MXF OP Atom files as well as the Source Pack treatments specified in **Table 2** of Section 4.5.3.3.2.

When the BodyUmid is newly created for the “Newly created” or “Replaced” in **Table 1**, it would have the same UMID value as the MpUmid according to Section 4.5.3.3.1. This is however problematic in the case of the MXF OP Atom files because of different MatTypes for the MpUmid and the BodyUmid. Since the MpUmid uniquely identifies the audiovisual material as a whole, its MatType must be “AV”. However, since the Edit Unit in question contains only the picture essence, the MatType of BodyUmid uniquely identifying the Edit Unit (with its zero Inst.#) must be “V”.

This contradiction comes from a singularity of the MXF OP Atom files that represents a single audiovisual material by using multiple MXF files containing a single essence. Therefore, an exceptional treatment of BodyUmid needs to be considered to address this issue.

One possible solution would be to prohibit the BodyUmid creation via the Generic Container in the MXF OP Atom file when it is newly created. Although this solution is complemented by the insertion of the MpUmid value at the playout of the MXF OP Atom files as shown in **Figure 34** (b), there is no placeholder to accommodate the Source Pack if it is also newly created during acquisition.

Another possible solution would be to assign the BodyUmid that inherits the Mat.# and the "AV" MatType from the MpUmid while its Inst.# being set to non-zero value even to the newly created Edit Unit. Note that, with this non-zero Inst.# value, this BodyUmid does not uniquely identifies the Edit Unit but just tells that the Edit Unit is related to the original material uniquely identified by the BodyUmid with its Inst.# masked to zero, though the original material in this case indicates itself containing the Edit Unit with the BodyUmid.

With this solution, although the origin of an Edit Unit no more exists from the viewpoint of BodyUmid as a unique material identifier of the Edit Unit, the BodyUmid is still used as a linking tool unambiguously referencing back to the original material that anyway contains the origin of the Edit Unit. Furthermore, because the BodyUmid is expected to take a form of the Extended UMID, it can accommodate the Source Pack even when it is newly created at acquisition, which is treated in the same way as shown in **Table 2** of Section 4.5.3.3.2.

In any way, a standard method on how to store the BodyUmid in the case of the MXF OP Atom files[38] needs to be explicitly specified by the relevant engineering document in SMPTE.

4.5.4.4 BodyUmid treatments for the AMWA AS-02 files

As introduced in Section 2.4.5, the AMWA AS-02 files are a set of MXF files, in which each mono type of essence is stored in an MXF OP1a single essence file called Essence Component file, while their synchronization as an audiovisual material is described by an MXF OP1b file called Version file, which externally references to the Essence Component files to be synchronized [11].

Thanks to the Essence Component file that is independent on its own, any Essence Component file in a given audiovisual material represented by the AMWA AS-02 files can be easily replaced with others, which enables easy repurposing of an MXF audiovisual program for multi-version, multilingual or multi-delivery of media environment.

At the time of this writing, there is no standard method on how to store the BodyUmid in the AMWA AS-02 files. When the audiovisual essence synchronization is taken into consideration, it would be ideally the Version file which ought to store a series of BodyUmids, each of which is assigned to an Edit Unit that is supposed to contain the synchronized picture and sound components. Another ideal option would be the provision of a dedicated Essence Component file that contains only metadata varying with the Edit Unit such as the BodyUmid as well as the timecode and technical metadata of a camera at acquisition.

On the other hand, because the Essence Component file itself is the MXF OP1a single essence file, the BodyUmid treatments discussed in Section 4.5.4.1.2 are applicable as they are. In other words, a series of BodyUmids for the AMWA AS-02 files can be stored with respective Edit Units in one of the Essence Component files containing a certain essence. In the following, assuming the BodyUmid to be stored in the Generic Container based, picture only, frame-wrapped Essence Component file, the BodyUmid treatments are discussed based on the creation of a set of AMWA AS-02 files from a series of incoming Material Units shown in **Figure 35** and **Figure 36**.

When a set of AMWA AS-02 files are newly created from scratch by, *e.g.*, acquisition (See Section 4.3.2.2.2), the MpUmid for each MXF file is also newly created. Note that because it is the Version file which represents a synchronized audiovisual material in MXF, the MpUmid which globally uniquely identifies the material composed of a set of AMWA AS-02 files as a whole needs to be contained in the Material Package of the Version file. Since each Essence Component file has its own newly created MpUmid whose MatType appropriately indicates the type of material contained in it, the Essence Component file in question (called picture Essence Component file, hereafter) has the MpUmid with the "V" MatType.

[38] Theoretically, each essence component in an Edit Unit can have its own BodyUmid. Hence the treatment of multiple BodyUmids per Edit Unit needs to be addressed if it is permitted.

The BodyUmid created in one of the ways shown in **Table 1** of Section 4.5.3.3.1 is then to be stored with the Edit Unit which contains only the picture essence as a frame in the Generic Container of the picture Essence Component file.

When the BodyUmid is created by the "Inherited" from the Extended UMID attached to an incoming Material Unit to be recorded as the Edit Unit in the picture Essence Component file as shown in the left-half of **Figure 36**, the BodyUmid is treated in the same way as discussed in Section 4.5.4.1.2, including the Source Pack treatments specified in **Table 2** of Section 4.5.3.3.2. Also, when the picture Essence Component file is played out on its own, the BodyUmid contained in it is transferred as is as shown in the right-half of **Figure 36**. Note that this BodyUmid transfer is applicable as is even when a set of AMWA AS-02 files is played out as an audiovisual material because the Material Unit supplied by the Version file is created from the corresponding Edit Unit with the BodyUmid being attached in the picture Essence Component file.

When the BodyUmid to be stored in the picture Essence Container is newly created for the "Newly created" or "Replaced" in **Table 1**, its UMID value must be equivalent with the MpUmid value globally uniquely identifying the picture Essence Component file as a whole as discussed in Section, which is then transferred as is at the playout of the picture Essence Component file alone. Note again that this BodyUmid transfer is applicable as is even when the set of AMWA AS-02 files is played out as an audiovisual material. This is rationalized by the reason also discussed in Section 4.5.4.1.2[39] while the open boxes drawn in red dashed line shown in the right-half of **Figure 35** are filled with the red because of the sound component(s) supplied from other Essence Component files containing the sound essence to be synchronized with the picture.

But what ought to be done when the MpUmid of the "AV" MatType, which globally uniquely identifies an audiovisual material as a whole, is desired to transfer at the playout of the material in a manner similar to that for the MXF OP1a AV interleaved file as discussed in Section 4.5.4.1.1, even when it is represented by the set of AMWA AS-02 files? This is resolved by the replacement of the basic part of BodyUmid with the MpUmid value contained in its Version file at its playout similar to that shown in **Figure 34** (b)[40]. Although this BodyUmid treatment requires additional processing for it, it is worthwhile to be done because it is the MpUmid of the Version file to be played out which globally uniquely identifies the resulting audiovisual material as a whole.

For example, when a new Version file is created by, *e.g.*, just replacing the Essence Component files of sound essence with others for its voice dubbed version, the newly created MpUmid for the Version file is more preferably transferred at its playout than the basic part of BodyUmid stored in the picture Essence Container file because it is the MpUmid which globally uniquely identifies the resulting voice dubbed version of audiovisual material as a whole.

Thanks to the BodyUmid to be stored in the picture Essence Component file in any case, the Source Pack can be accomodated regardless of the BodyUmid treatment shown in **Table 1** of Section 4.5.3.3.1, which is treated in the same way as specified in **Table 2** of Section 4.5.3.3.2.

In any way, a standard method on how to store the BodyUmid in the case of the AMWA AS-02 files[41] needs to be explicitly specified by the relevant engineering document in SMPTE or elsewhere.

---

[39] Even when a UMID attached to the Material Unit with the "V" MatType is used to create a new BodyUmid by the "Inherited" at the recorder side, the logically expected functions of BodyUmid discussed in Section 4.5.3.2 is realized, though it is not the Version File but the picture Essence Component file containing the Edit Unit of picture only, which is referenced back by the BodyUmid with its Inst.# masked to zero.

[40] The BodyUmid to be stored in the picture Essence Component file would have been created from the MpUmid of the original Version file with its Inst.# set to non-zero if we follow the BodyUmid creation proposed for the MXF OP Atom files in Section 4.5.4.3. This proposal, however, ought to be avoided for the AMWA AS-02 files. This is because of the BodyUmid transfer at the playout of another Version file that references to the same picture Essence Component file with different sounds. Although it is logically valid for the BodyUmid created based on the proposal to be used to reference back to the original Version file, it is the other Version file actually played out which is more preferably referenced buck by the BodyUmid in practice, which is achieved by replacing the basic part of BodyUmid with the MpUmid of the other Version file at its playout.

[41] Theoretically, each Essence Component file can contain its own BodyUmid in it. Hence the treatment of multiple BodyUmids per Edit Unit needs to be addressed if it is permitted.

5 Examples of UMID Applications in MXF and Streaming Media

5.1 Introduction

In this section, some plausible UMID applications specifically for the MXF technology in the real world are demonstrated by considering all the UMID treatments discussed in Section 4.

5.2 Some constraints assumed for the demonstrations

To simplify the matter, the following constraints are assumed for all the UMID applications in MXF and the streaming media demonstrated in this section.

(a) All the MXF files to be treated in the demonstrations are based on the AV interleaved, frame-wrapped MXF OP1a files (See Section 4.5.4.1.1) in which each frame is lossy compressed. Consequently,

- Each Edit Unit in the MXF file is composed of a (lossy compressed) frame and all the sound chunks that synchronize with the frame, to which a single BodyUmid of its MatType being "AV" (the byte 11th of the UMID UL set to "0D$_h$") is assigned,
- A new or updated BodyUmid is assigned to the Edit Unit at its creation, and it is embedded as is to a designated position at the Edit Unit transfer, including its playout as a part of the baseband audiovisual stream (See **Figure 34** (a) in Section 4.5.3.5).

Note that this constraint does not exclude the use of other MXF OPs as a recording format of the material. In other words, as long as it is regarded as an AV interleaved, frame-wrapped MXF file from an external application viewpoint, any other MXF OPs such as the MXF OP Atom or the AMWA AS-02 are acceptable as the actual recording format. Please visit Sections 4.5.4.3 and 4.5.4.4 for the BodyUmid treatments in the MXF OP Atom files and the AMWA AS-02 files, respectively.

(b) Each MXF file is globally uniquely identified on its own. Consequently,

- A newly created MpUmid of a new Mat.# and zero Inst.# values is assigned at the new MXF file creation.

(c) The FpUmid of an MXF file is used as a linking tool referencing back to a source from which the MXF file is created. Consequently,

- The FpUmid of the Mat.# value inherited from the UMID globally uniquely identifying the source and non-zero Inst.# value is assigned at the new MXF file creation.

(d) The Material Unit to be treated in the demonstrations is composed of a baseband frame and all the sound chunks that synchronize with the frame. Consequently,

- When an Edit Unit is (decompressed and) placed on the wire for its transfer, it corresponds to the Material Unit in question one by one. In other words, a frame with its synchronized sound chunks is called an Edit Unit when it is recorded as a part of the MXF file, and it is called a Material Unit when it is placed on the wire for its transfer[42],
- The BodyUmid assigned to an Edit Unit corresponds to a UMID attached to the corresponding Material Unit, both of which share the MatType as "AV".

(e) Both the BodyUmid and the UMID attached to the Material Unit take a form of the Extended UMID. Furthermore,

- In principle, the Source Pack initially created at the original creation of either the Material Unit or the Edit Unit is always inherited as is, resulting in that the Source Pack treatment is sometimes omitted in the demonstrations when it is self-evident.

(f) The basic part of BodyUmid is either newly created or created by inheriting the UMID attached to an incoming Material Unit, in which the new non-zero Inst.# value for the BodyUmid is created based on the "Copy number

[42] A frame in the Edit Unit is compressed while that in the corresponding Material Unit is uncompressed. They are however identical in the sense of the UMID Application Principle 4 (UMID Identification).

and 16-bit PRS (Pseudo-Random Sequence) generator" method specified in SMPTE ST 330 [1] with the bottom nibble of the byte 12th of UMID UL set to "3_h" for the latter case. Furthermore,

- Even when a certain media processing such as a transcoding, a media filter or an image/text overlay is applied to a Material Unit on the wire before it is recorded, the copy number in the Inst.# field of the UMID attached to the Material Unit is employed, *i.e.*, it is incremented by one per a single media processing, even when the bottom nibble of the byte 12th of UMID UL remains unchanged as "F_h"[43].

Note that only the copy number is indicated when an Inst.# value is shown in the figures in this section. As a result, even when two Inst.# values appearing in the figure are same, this does not always mean that the entire Inst.# values including their PRS numbers are identical. Rather, they ought to be considered to have different PRS numbers in general unless the identicalness of those Inst.# values is explicitly mentioned in the text.

(g) The Mat.# value is always created based on the "SMPTE method" specified in SMPTE ST 330 [1] with the top nibble of the byte 12th of UMID UL set to "1_h". Consequently,

- The byte 12th of the BodyUmid is always set to "13_h".

(h) The UMID "Live stream" flag (the bottom nibble of the byte 12th of UMID being set to "F_h") is employed for the UMID attached to the Material Unit which has never been recorded. Consequently,

- The byte 12th of the UMID attached to the Material Unit which has never been recorded since its original creation is always set to "$1F_h$".

Note that the bottom nibble of the byte 12th of BodyUmid cannot take the UMID "Live stream" flag but is always set to "3_h" because of the Edit Unit as the recorded one. In other words, when a Material Unit on the wire with the UMID "Live stream" flag is recorded as the Edit Unit, the byte 12th of an incoming UMID as "$1F_h$" must be always changed to "13_h" for the BodyUmid to be assigned to the Edit Unit.

Furthermore, the UMID "Live stream" flag for the MpUmid is not employed in the demonstrations. In other words, the treatment of the growing MXF file is not considered here for simplicity. As it is orthogonal to the UMID applications in MXF and the streaming media demonstrated in this section, however, it can be additionally applied in the same way as discussed in Section 4.3.4 when needed.

(i) When an MXF recorder newly creates an MXF file, it applies a single BodyUmid treatment to all the Edit Units contained in the MXF file being created. Consequently,

- When the UMID inheritance is applied, it is effective only to the BodyUmid that has an incoming UMID to be inherited. Otherwise, any BodyUmid treatment including a new BodyUmid creation must not be applied without the incoming UMID,
- When the assignment of, or the replacement with, a newly created UMID is applied, the basic part of BodyUmid to be assigned to all the Edit Units in the MXF file being created must be equivalent with the newly created MpUmid uniquely identifying the MXF file as a whole,
- In either case, the Source Pack is newly created only when there is no Source Pack to be inherited and the basic part of BodyUmid is newly created. Otherwise, a default action to inherit the existing Source Pack as is must be always taken.

Note that these constraints are also applied to the treatment of UMID attached to the Material Unit on the wire when it is processed in a certain way, *i.e.*, a single UMID treatment must be applied to all the Material Units in the media stream during when a certain media processing is applied to the stream.

(j) When a camera newly creates and transfers a sequence of Material Units on the wire, it attaches a constant basic part of UMID with the "Live stream" flag (composed of a constant Mat.# and zero Inst.# values uniquely identifying the camera) and the Source Pack (varying over the Material Units) to every Material Unit it creates. Consequently,

[43] To be strict, because the copy number is employed only when the bottom nibble of the byte 12th of UMID UL is set to "3_h" according to SMPTE ST 330, its employment is incompatible with the UMID "Live stream" flag that requires the bottom nibble of the byte 12th to be set to "F_h". A future ST 330 revision needs to address this problem.

- The byte 12th of the UMID is always set to "1F$_h$".

Note that the constant basic part of UMID that contains a constant Mat.# value regardless of the operational status of a camera can be obtained by using either a manually pre-set time stamp value such as the UNIX Epoch time (1970-01-01T00:00:00Z) or a time stamp at which a camera is initially powered on for the Mat.# creation.

5.3 Example 1 – An original MXF file creation by the live feed capture

5.3.1 Live feed without UMID

Figure 37 schematically demonstrates one of the most fundamental examples of UMID applications in MXF, *i.e.*, an original MXF file creation by capturing the live feed from a camera. In this example, the camera ("Camera 0") creates and transfers a sequence of Material Units on the wire (*e.g.*, SDI) without UMID being attached, which is then received by an MXF recorder ("MXF Recorder 1") and recorded to create an MXF file ("Source0.mxf").

This system configuration is very common when the media products commercially available today in the industry are used. Furthermore, this example also represents the cases of a camcorder, or a vendor-specific system in which the camera and the MXF recorder is connected by using a proprietary technology.

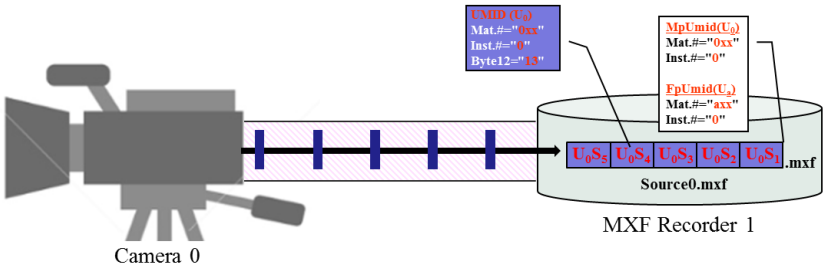


Figure 37: A new MXF file creation from the live feed without UMID

In this example, because of an original MXF file creation, the MpUmid ("U_0") uniquely identifying the MXF file as a whole must be newly created with a new Mat.# ("0xx") and zero Inst.# values. Both the FpUmid ("U_a") and all the BodyUmids are also newly created because of absence of the "source" UMID to be inherited, in which the basic part of BodyUmid must be equivalent with the MpUmid as "U_0" according to Section 4.5.3.3.1.

While it is obvious that the Source Pack ("S_j", j=1, 2 …) is also newly created, how can we obtain the information to create it? In the case of a camcorder that integrates the camera and the MXF recorder in a single body, a clock and the GPS (Global Positioning System) unit mounted on it are to be used to obtain the "When" and "Where" information, while a preset value is used for the "Who" information[44].

When a camera is separated from a recorder and they are located at completely different places, a clock mounted on the MXF recorder is to be used for the "When" information. Similar to the "Who" information, a value can be pre-set at the MXF recorder to provide the "Where" information if the camera is located at a fixed place known in advance. Furthermore, there is another option that the location of the MXF recorder instead of that of the camera is used for the "Where" information[45].

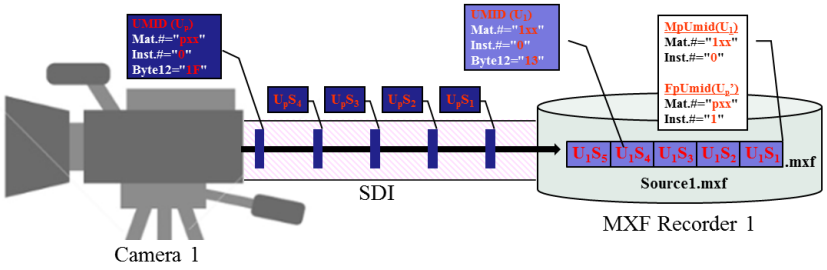
5.3.2 Live feed with UMID

If a camera has a capability to generate a constant basic part of UMID (with a variable Source Pack being appended) and to attach it to the Material Units it continuously creates, there are two options for the MXF recorder to create the BodyUmid to be stored within the MXF file being created. **Figure 38** (a) and (b) below schematically demonstrate the cases, in which a camera ("Camera 1") creates and transfers a sequence of Material Units on the wire (SDI) with the

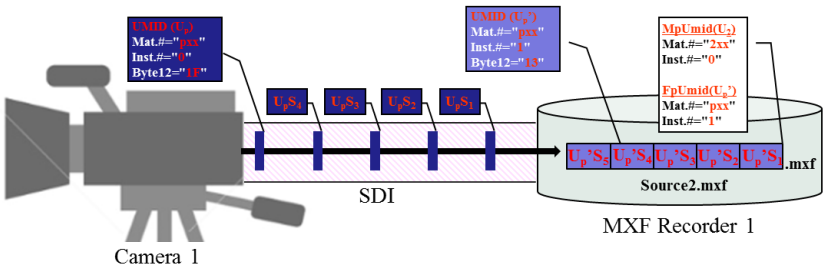
[44] See **Figure 14** for the Source Pack structure.

[45] According to SMPTE ST 330 [1], the top nibble of the byte 4th of the "Where" field (or the nibble 7 of the "Altitude" field) is used to indicate what location information it is. For example, the values of "A$_h$" and "B$_h$" for the nibble are defined for the location of the sensor device and the recording device, respectively.

Extended UMID having the constant basic part ("U_p") being attached[46]. The sequence of Material Units is then received by an MXF recorder ("MXF Recorder 1") and recorded to create either one MXF file ("Source1.mxf") with the newly created BodyUmid ("U_1") for **Figure 38** (a) or another MXF file ("Source2.mxf") with the inherited BodyUmid ("U_p' ") for **Figure 38** (b).



(a) Recorded with the newly created BodyUmid



(b) Recorded with the inherited BodyUmid

Figure 38: A new MXF file creation from the live feed with UMID

Because an original MXF file is created in both cases anyway, the MpUmid ("U_1" or "U_2") uniquely identifying the MXF file as a whole must be newly created with a new Mat.# ("1xx" or "2xx") and zero Inst.# values[47].

Thanks to the constant value for the basic part of UMID attached to all the Material Units to be recorded in both cases, even for an MXF OP1a file creation, the FpUmid can be created by inheriting the (basic part of Extended) UMID attached to an incoming Material Unit with the inherited Mat.# ("pxx") and a newly created non-zero Inst.# ("1") values, though the UMID "Live stream" flag ("F_h") needs to be replaced with the number indicating the Instance Number generation method ("3_h" in the case of the "Copy number and 16-bit PRS generator" method assumed in the demonstrations) as discussed in Section 4.4.3.5[48].

As for the BodyUmid, when it is newly created as shown in **Figure 38** (a), its basic part must be equivalent with the MpUmid value. Note that this is the case in which the Edit Unit is regarded as the original material at its point of entry into the UMID based material management, and therefore, the BodyUmid treatment in **Figure 38** (a) is identical with that shown in **Figure 37** regardless of an existence of the "source" UMID to be inherited.

In addition, this case is also effective even when the material "source" is not a camera but an MXF player for an already recorded MXF file to be played out with the BodyUmids contained in the MXF file. In other word, an Edit Unit

[46] To be strict, it is the basic part of the Extended UMIDs being attached to the Material Units which is constant, and (at least the "When" field of) the Source Pack of the UMIDs varies depending on the Material Units.

[47] For simplicity, different MpUmids are assigned to the MXF files shown in **Figure 37**, **Figure 38** (a) and (b). If the resulting MXF files are identical in the sense of the UMID Application Principle 4 (UMID Identification) but just have different FpUmids and/or BodyUmids only, they can share a single MpUmid. Note that this does not apply to the case of multiple recordings from a single source in general, which is further discussed in Section 5.6.

[48] For some reasons, there would be a case in which the FpUmid is intentionally newly created with a new Mat.# and zero Inst.# values. This is in fact only the option to be taken if a single "source" UMID for the UMID inheritance is unknown or cannot be uniquely determined in the case of the MXF OP1a file creation. See Section 5.4 for more discussion about the FpUmid creation.

assigned with a newly created BodyUmid with its zero Inst.# is always regarded as the original *Material* from which any *Instance* can derive[49] regardless of its material "source" kinds, according to the UMID Application Principle 2 (UMID Creation).

When it is created by inheriting the UMID attached to an incoming Material Unit as shown in **Figure 38** (b), on the other hand, its basic part can be equivalent with the FpUmid value[50].

Note that in both cases the Source Pack must be inherited as is. Because SMPTE ST 291 [19] has already specified how to deliver the Extended UMID via SDI, the "When", "Where" and "Who" information a camera obtains during the acquisition by using *e.g.*, its internal clock and the GPS unit mounted on it, can be transferred and stored as is into an MXF file being created even based only on the SMPTE standard technologies, which will be the most apparent benefit for the camera to be encouraged to support the UMID applications[51].

5.3.3 Is BodyUmid to be newly created or to be inherited?

Because an Edit Unit is assumed to contain a lossy compressed frame in the demonstrations, it is not identical with a corresponding Material Unit a camera initially creates and transfers on the wire in the sense of the UMID Application Principle 4 (UMID Identification). If we strictly follow this observation, only the Material Unit on the wire would have been regarded as the true original material, *i.e.*, the BodyUmid to be assigned to the Edit Unit has to be always created by the UMID inheritance as shown in **Figure 38** (b).

There is however a serious disadvantage for such a BodyUmid creation, *i.e.*, because the said original material only temporarily exists on the wire in this BodyUmid creation, there is no material to be referenced back as the original one even when the UMID assigned to the derived material is to be resolved by using the standard UMID resolution.

Considering that one of the main advantages of the UMID application is the UMID propagation (more strictly, a propagation of the Mat.# of the original material) over the media production workflow chain so that the origin of any material within the workflow chain can be unambiguously identified, it is highly desired for the original material to persistently exist somewhere within a media production system.

Furthermore, the assumption that a camera always attaches a UMID having a constant basic part to the Material Units it creates regardless of its operational status prevents different bounded sequences of Material Units created by the camera at different times from being distinct by the UMID. In fact, it is the MXF recorder which determines the boundaries of the Material Unit sequence by its REC Start and Stop operations, which corresponds to the Case A discussed in Section 4.5.2.3.2, or the material dubbing as new materials, while the source material in **Figure 27** of the section corresponds to an open-end single sequence of Material Units uniquely identified by the UMID.

Therefore, the BodyUmid is recommended to be newly created as shown in **Figure 38** (a) unless otherwise explicitly specified for some special reasons.

It is interesting to note that if a camera attaches different UMIDs to the Material Units in different sequences and every Material Unit is recorded as the Edit Unit exactly as is or in a lossless compressed fashion, a UMID attached to the Material Unit in a sequence is theoretically reused as the BodyUmid exactly as is because of an identicalness of the Edit Unit with the corresponding Material Unit according to the UMID Application Principle 4 (UMID Identification).

This is in fact how a camcorder behaves when it is observed that its camera part creates a Material Units having a lossy compressed frame (instead of a baseband frame) with such a UMID being attached, and its MXF recorder part records the Material Unit as the Edit Unit exactly as is including its UMID as the BodyUmid[52] (and then assigns the MpUmid equivalent with the basic part of the BodyUmid to uniquely identify the resulting MXF file as a whole).

---

[49] The UMID Application Principle 1 (Definitions) defines the precise meanings of *Material* and *Instance* used in the statement.

[50] An advantage to create a BodyUmid fully equivalent with the FpUmid including its PRS number is worthwhile to be explored.

[51] A similar benefit would have been obtained even for a proprietary technology to be used between a camera and an MXF recorder if it supports the Source Pack equivalent information delivery. In this case, however, a user cannot choose the MXF recorder from a different vendor from the one that supplies the camera.

[52] Because the Mat.# in the BodyUmid varies with bounded sequences of Material Units in this case, it cannot be used as the material "source" identifier in a way as discussed in Section 4.4.3.5 in general. In fact, because the Mat.# is often created as a

5.4 Example 2 – An MXF file creation by partial retrieval of an existing MXF file

5.4.1 Partial retrieval within a device

Figure 39 schematically demonstrates an example in which a part of an existing source MXF file (“Source1.mxf”) is specified by its In/Out points and retrieved to form a resulting MXF file (“Result1.mxf”) within an MXF player (“MXF Player 1”), which is then transferred to the outside of the MXF player. Note that this example is a comprehensive version of the one shown in **Figure 19** additionally including the BodyUmid treatment.

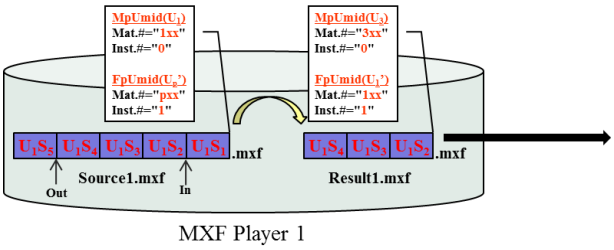


Figure 39: Partial retrival of an existing MXF file wihin a device

In **Figure 39**, because the resulting MXF file is a newly created one to be managed independently, its MpUmid (“U_3”) uniquely identifying the MXF file as a whole is newly created with a new Mat.# (“3xx”) and zero Inst.# values.

As already demonstrated in **Figure 19**, the FpUmid of the resulting MXF file is created by inheriting the MpUmid of the source MXF file, *i.e.*, composed of the inherited Mat.# (“1xx”) and non-zero Inst.**#** (“1”) values. Note that behind this FpUmid creation, there is an assumption that the MpUmid of the source MXF file is known in advance, and this assumption usually holds valid in this case because all the MXF files within a device are managed based on their MpUmids as a globally unique material identifier.

As for the BodyUmid of the resulting MXF file, because the Edit Units within the source MXF file are to be reused exactly as they are in the resulting MXF file, the BodyUmids attached to them are also the same including its Source Pack, indicating that the Edit Units in the source MXF file specified for the partial retrieval are cloned to those in the resulting MXF file including their BodyUmids.

It is worthwhile to note that because a valid MpUmid in the sense of the UMID Application Principle 3 (UMID Integrity) is always assigned to the resulting MXF file in this example, a client that requests and receives the resulting MXF file from the MXF player needs not care about its validity. In other words, even if the resulting MXF is retrieved by using a generic “UMID-unaware” file retriever such as a generic FTP client (with an extended function to specify the In/Out points), a user can trust the validity of the MpUmid and thus use it as is for his/her UMID based material management as needed.

When multiple partial retrievals with the same In/Out points are requested, it is ideally expected that an MXF player supplies an identical resulting MXF file including its MpUmid. This implies that the MXF player is anticipated to support the history management of the partial retrievals, *i.e.*, every partial retrieval request successfully conducted is recorded with its source MXF file, the specified In/Out points and the MpUmid created and assigned to the resulting MXF file at its first request, and an identical resulting MXF file is reproduced when an already conducted request in the past is given again, though different MpUmid assignment to the resulting MXF file for every identical requests are still acceptable from the viewpoint of the UMID Application Principle 3 (UMID Integrity).

5.4.2 Partial retrieval over the network

A similar behavior to that discussed in Section 5.4.1 is observed for the partial retrieval over the network when the retrieved Edit Units are recorded exactly as they are at the receiver side. **Figure 40** schematically demonstrates such an example, in which a part of an existing source MXF file (“Source1.mxf”) in an MXF player (“MXF Player 2”) is requested to retrieve with its In/Out points, and the Edit Units contained in the specified part of the source MXF file

---

combination of the time stamp and the device node number uniquely identifying the material source such as a camera, an analysis of the Mat.# might be able to reveal the device node number as the material “source” identifier, though such a Mat.# parse is usually discouraged because of its recommended treatment as a single dumb number (See Section 5.3 in [2]).

are transferred as a sequence of the Material Units over the network, and recorded as the Edit Units exactly as they are at an MXF recorder (“MXF Recorder 1”) to form a resulting MXF file (“Result2.mxf”).

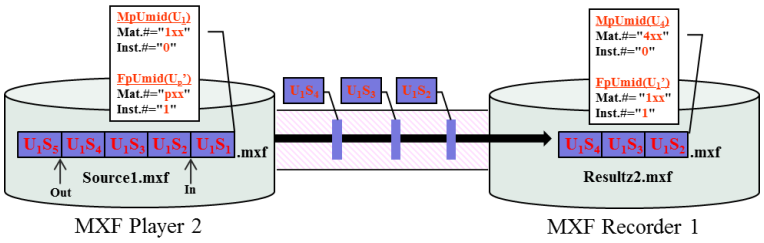


Figure 40: Partial retrival of an existing MXF file over the network

In **Figure 40**, because the resulting MXF file is a newly created one to be managed independently, with the Edit Units identical with those in the source MXF file in the sense of the UMID Application Principle 4 (UMID Identification), the MpUmid and the BodyUmids are newly created and reused, respectively, as already demonstrated in Section 5.4.1.

As for the FpUmid of the resulting MXF file, on the other hand, it would be created in the same way as discussed in Section 5.4.1 if the MpUmid of the source MXF file is known in advance, which is achieved when the source MXF file is specified by its MpUmid. But what happens if the MpUmid of the source MXF file is unknown because of the source MXF file being specified by using its URL rather than its MpUmid?

When the MpUmid to be inherited by the FpUmid of the resulting MXF file is unknown or cannot be determined uniquely, there is no choice but to newly create the FpUmid with a new Mat.# and zero Inst.# values in principle, which in fact constitutes the default behavior of the FpUmid creation as discussed in Section 4.4.2.

Note that even if this is the case, the original material can be referenced back when the BodyUmid assigned to the Edit Units in the resulting MXF file is to be resolved, though the original material in this case is not always the source MXF file if, for example, the source MXF file by itself is the one created by the partial retrieval, resulting in that the basic part of its BodyUmid is not equivalent with its MpUmid (See Section 5.5.2 for more discussion about it).

Resembling to the case discussed in Section 4.4.3.4, if all the UMIDs attached to the incoming Material Units during recording are identical, the FpUmid of the resulting MXF file is created by inheriting the UMID value. More specifically, an anticipated behavior of the MXF recorder include

1) the MXF recorder temporarily creates the initial FpUmid of the resulting MXF file by inheriting the UMID attached to the first Material Unit it receives,

2) the MXF recorder continuously monitors the value of UMID attached to subsequent incoming Material Units until the end of recording,

3) the MXF recorder replaces the initially created FpUmid value with the newly created one if it detects different UMID value from that attached to the first Material Unit during the recording,

though the original material to be referenced back by the resolution of the FpUmid created in this way is again not always the source MXF file as discussed above for the BodyUmid resolution.

5.5 Example 3 – An MXF file creation by partial recording of an MXF file playout

5.5.1 For the original MXF file playout

Figure 41 on the next page schematically demonstrates an example in which a resulting MXF file (“Result3.mxf”) in an MXF recorder (“MXF Recorder 1”) is newly created by partially recording a playout (over the SDI) result of an existing source MXF file (“Source1.mxf”) by an MXF player (“MXF Player 2”), where the source MXF file is the original one newly created by, *e.g.*, the live feed capture as demonstrated in Section 5.3, resulting in its MpUmid and the basic part of its BodyUmid being equivalent.

Although this example looks like the one demonstrated in **Figure 40** at a glance, there is a substantial difference between them. While the MXF player partially retrieves the Edit Units from the source MXF file in it and transfers them to the MXF recorder as a bounded sequence of the Material Units in **Figure 40**, it is the MXF recorder which

determines the boundaries of the sequence of Material Units to be recorded as the Edit Units to form the resulting MXF file within an incoming sequence of the Material Units transferred from the MXF player in this example[53].

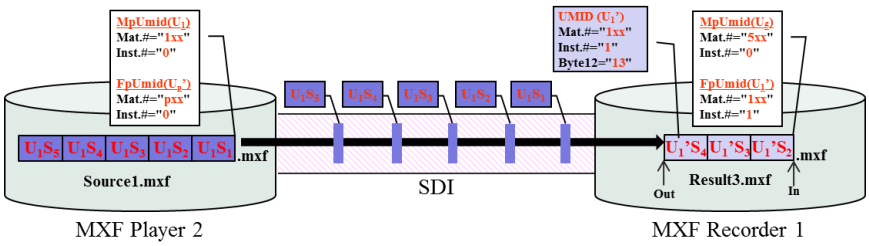


Figure 41: An MXF file creation by partial recording of the original MXF file plaout

In addition, while an incoming Material Unit transferred over the network is assumed to be recorded exactly as is as the Edit Unit including the UMID attached to the Material Unit as the BodyUmid in **Figure 40**, the Material Unit over the SDI is recorded as the Edit Unit in a lossy compressed fashion in this example, which results in that the BodyUmid to be assigned to the Edit Unit is created by inheriting the UMID attached to an incoming Material Unit[54].

In **Figure 41**, because the resulting MXF file is a newly created one to be managed independently, its MpUmid ("U_5") uniquely identifying the MXF file as a whole is newly created with a new Mat.# ("5xx") and zero Inst.# values.

Like the example in **Figure 40**, the FpUmid of the resulting MXF file is created in the same way as discussed in Section 5.4.2 if the MpUmid of the source MXF file is known in advance. The FpUmid creation at the unknown MpUmit to be inherited discussed in that section is also applicable to this example exactly as is.

As for the BodyUmid of the resulting MXF file, as already mentioned, it is created by inheriting an incoming UMID attached to the corresponding Material Unit, *i.e.*, with the inherited Mat.# ("1xx") and the non-zero Inst.**#** ("1") values. Note that there is another option that the BodyUmid is to be newly created with a new Mat.# equivalent with the MpUmid ("U_5") and zero Inst.# values for some reasons. But which of the BodyUmid creations ought to be more encouraged in this case?

In Section 5.3.3, it is recommended that in the case of the original MXF file creation, the BodyUmid assigned to the Edit Units contained in it ought to be newly created as shown in **Figure 38** (a) in order for a descendent Edit Unit to be contained in any material derived from the original MXF file in any way to be able to reference back to the original MXF file when its BodyUmid is resolved by using the standard UMID resolution.

In this case, however, with a persistent existence of the original material to be referenced back, it is obvious that the BodyUmid ought to be created by the UMID inheritance in order to propagate the Mat.# of the original material over the media production workflow chain so that the origin of any material within the workflow chain can be uniquely identified without ambiguity.

It is worthwhile to note that the original MXF file is also defined as the one in which the basic part of its BodyUmid is always equivalent with its MpUmid. When the source MXF file is the original MXF file as shown in **Figure 41**, both the FpUmid and the BodyUmid of the resulting MXF file are therefore to be resolved to the same one, *i.e.*, the source MXF file which is regarded as the original MXF file.

As for the Source Pack treatment, it must be inherited as is in principle especially when the BodyUmid is created by the UMID inheritance though there still remains an option for the Source Pack to be deleted for security reasons.

---

[53] This is in fact equivalent with the one demonstrated in **Figure 38** as far as the MXF recorder's behavior is concerned.

[54] If a Material Unit is transferred in a compressed fashion such as over the SDTI-CP [16] and stored as the Edit Unit exactly as is, the UMID attached to the Material Unit can be also reused for the BodyUmid to be assigned to the Edit Unit.

5.5.2 For the derived MXF file playout

When the source MXF file to be played out is not the original MXF file but the derived one, a caution ought to be taken for the material to be referenced back from the resulting MXF file to be obtained by using the UMID resolution.

Figure 42 schematically demonstrates an example in which the resulting MXF file obtained in the example shown in **Figure 41** ("Result3.mxf") is further played out as a source MXF file, and a new resulting MXF file ("Result4.mxf") is created in an MXF recorder ("MXF Recorder 1") by partially recording the playout result of the source MXF file by an MXF player ("MXF Player 2").

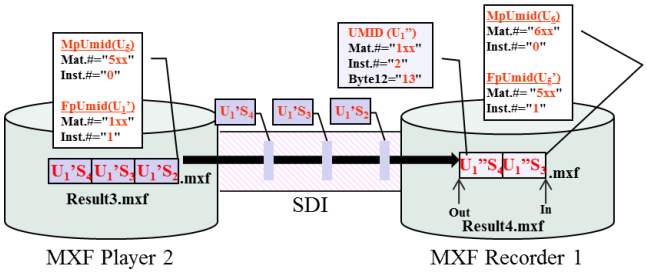


Figure 42: An MXF file creation by partial recording of the derived MXF file playout

Because only a difference between the examples shown in **Figure 41** and **Figure 42** is the source MXF file to be played out, the MXF recorders in both examples behave in the same way. Consequently, both the MpUmid and the FpUmid for the resulting MXF file in **Figure 42** are created in the same way as for the example shown in **Figure 41**.

The BodyUmid of the resulting MXF file is also created by the UMID inheritance including its Source Pack in this example. Because the UMID attached to an incoming Material Unit was also created by the UMID inheritance from the original MXF file, this means that the UMID inheritance has been applied to the original BodyUmid twice to create the BodyUmid to be assigned to the Edit Unit in the resulting MXF file. As a result, the copy number in its Inst.# field is set to "2" as shown in **Figure 42**, which also indicates that a certain media processing (the lossy media compression in this case) has been applied to the Edit Unit twice since its origin.

It ought to be noted that because the basic part of BodyUmid is not equivalent with the MpUmid in the derived MXF file such as the source MXF file ("Result3.mxf") shown in **Figure 42**, the resolution of the FpUmid differs from that of the BodyUmid for the resulting MXF file ("Result4.mxf"), *i.e.*, while the FpUmid of the resulting MXF file is resolved to the source MXF file ("Result3.mxf" in **Figure 42**), the BodyUmid of the resulting MXF file is resolved to the original MXF file ("Result1.mxf" in **Figure 41**) from which the source MXF file derives.

5.6 Example 4 – Multiple MXF file creations from a single source

5.6.1 Via a source splitter

Figure 43 schematically demonstrates an example in which a live feed from a camera ("Camera 1") is split into two streams, which are captured by two MXF recorders ("MXF Recorder 1" and "MXF Recorder 2") to create respective MXF files ("Source1.mxf" and "Source3.mxf").

While the same In/Out points are assumed to be specified by both the MXF recorders in this example, each MXF recorder can generally independently specify the In/Out points at its discretion. Furthermore, each MXF recorder can adopt its own codec and/or different technical parameters even in the same codec. The Edit Units being created by those MXF recorders are therefore not identical in the sense of the UMID Application Principle 4 (UMID Identification) in general even when they are simultaneously created from the same sequence of Material Units. Consequently, each MXF file is assigned with its own newly created MpUmid ("U_1" and "U_7") and the BodyUmids whose basic parts are equivalent with the respective MpUmids for the example shown in **Figure 43** (See next page).

As for the FpUmids of those MXF files, their sharing the same source is indicated by their FpUmids ("U_p' ") created by the UMID inheritance with the inherited Mat.# ("pxx") and the non-zero Inst.# ("1") values as shown in **Figure 43**[55]. Furthermore, the Edit Units in those MXF files derived from the same Material Unit share the same Source Pack (*e.g.*, "S_1") which tells "When", "Where" and "Who" has originally created the Material Unit. Hence, the combination of the FpUmid (with its Inst.# masked to zero) and the Source Pack can unambiguously identify the original Material Unit from which those Edit Units actually derive[56].

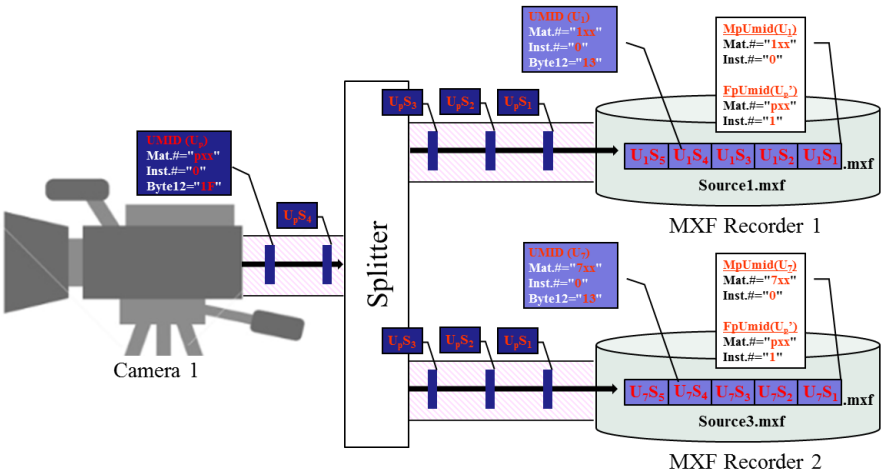


Figure 43: Multiple recording from a single source via a source splitter

So what ought to be done if one desires to obtain more than one identical MXF file? One obvious method would be to create the clone MXF files including their MpUmids. But is there any other way to create them all at once?

Theoretically, if multiple MXF recorders can simultaneously create the MXF files with the same In/Out points and the corresponding Edit Units having exactly the same bit pattern, those MXF files would be regarded as identical in the sense of the UMID Application Principle 4 (UMID Identification), and therefore, they could share the same MpUmid.

In reality, however, it is hard to be achieved because there is no guarantee for different MXF recorders to create identical MXF files in the sense of the UMID Application Principle 4 (UMID Identification), *i.e.*, even if they adopt the same codec with the same technical parameters, their compression algorithms and/or their implementations of the algorithms might be slightly different, resulting in giving different bit patterns to the Edit Units they create.

One possible solution might be to use an identical product model for the MXF recorders and they are synchronously controlled. Even in this case, however, an unexpected disturbance happened to an MXF recorder might differentiate the bit patterns being created for the Edit Units. Hence, their identicalness needs to be verified after their creations by using such as the comparison of their hash values[57] if they are desired to be assigned with a single MpUmid[58].

---

[55] To signal simultaneous recording of those MXF files, the use of a predetermined value instead of the PRS (Pseudo-Random Sequence) number for the last two bytes of the Inst.# field might be functional though this requires a new Inst.# generation method to be additionally specified in SMPTE ST 330 [1].

[56] There is in fact no original Material Unit to be referenced back in this example because it only temporarily exists on the wire. This FpUmid and Source Pack treatment is more effective when an original MXF file that exists persistently elsewhere is played out as a material source.

[57] Because the Mat.# generation method assumed in this demonstration, or the "SMPTE Method" [1], always gives different UMID values even for the MXF recorders of an identical product model at the same time point, the values in the corresponding UMID related fields in the MXF files being created are always different, which ought to be ignored for the comparison of those MXF files.

[58] If those MXF files are found not to be identical, not only the MpUmid but also the BodyUmids assigned to individual Edit Units in one of those MXF files are required to be replaced with a newly created UMID value unless the BodyUmids are created by the UMID inheritance..

5.6.2 Via a read-while-write MXF recorder/player

Some MXF recorders have a capability to play out an MXF file still under creation. **Figure 44** on the next page schematically demonstrates such an example in which a live feed from a camera ("Camera 1") is captured by using the recording capability of an MXF recorder/player ("MXF Recorder/Player") to create an original MXF file ("Source1.mxf"). The Edit Unit in the original MXF file is then played out immediately after it is created by using the playout capability of the same MXF recorder/player and it is further captured by another MXF recorder ("MXF Recorder 1") to create a derived MXF file ("Result5.mxf").

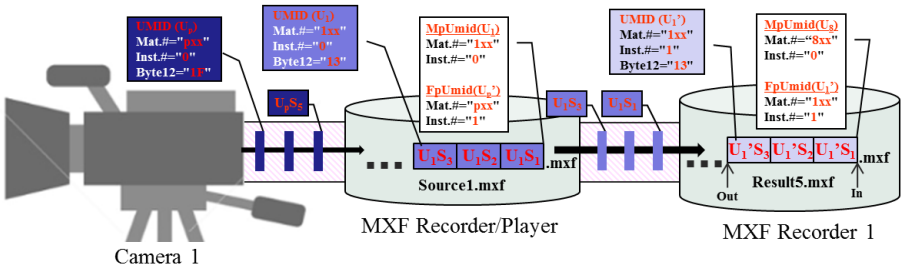


Figure 44: Multiple recording via a read-while-write MXF recorder/player

From the UMID application viewpoint, whether those two MXF files are created in parallel as shown in **Figure 44** or they are created one by one in such a way shown in **Figure 38** (for the original MXF file creation) followed by that in **Figure 41** (for the derived MXF file creation) does not differentiate those resulting MXF files. In fact, the creations of the original and the derived MXF files shown in **Figure 44** ("Source1.mxf" and "Result5.mxf") are equivalent with those shown in **Figure 38** (a) ("Source1.mxf") and in **Figure 41** ("Result3.mxf")[59], respectively.

As a result, each MXF file has its own newly created UMID value ("U_1" and "U_8" for "Source1.mxf" and "Result5.mxf", respectively)[60]. Their FpUmids are created by the UMID inheritance with their respective inherited Mat.# ("pxx" and "1xx") and the non-zero Inst.# ("1") values.

As for the BodyUmid of those MXF files, while the BodyUmid whose value is equivalent with the newly created MpUmid is assigned to the Edit Unit in the original MXF file ("Source1.mxf"), the BodyUmid created by inheriting the UMID attached to an incoming Material Unit is assigned to the Edit Unit in the derived MXF file ("Result5.mxf"), as discussed in Sections 5.3.3 and 5.5.1, respectively.

5.7 Example 5 – A single MXF file creation from multiple sources

5.7.1 From two cameras as sources

Figure 45 on the next page schematically demonstrates an example in which two live feeds from the respective cameras ("Camera 1" and "Camera 2") are supplied and switched over at a switching device ("Switcher") to form a single stream, which is then captured by an MXF recorder ("MXF Recorder 1") to create an original MXF file ("Result6.mxf").

In this example, since a transition effect such as a wipe is applied to the switching point over two frames, the Material Units around the point are processed in a certain way by the switching device, while those in other parts go through it as they are without any influence.

In **Figure 45**, because of the original MXF file creation, the MpUmid ("U_9") uniquely identifying it as a whole is newly created with a new Mat.# ("9xx") and zero Inst.# values. The basic part of its BodyUmid is created as equivalent with the MpUmid including its UMID "Live stream" flag reset from "F_h" to "3_h", while the Source Pack appended to the incoming UMID is always inherited as is.

---

[59] While a part of incoming Material Units are recorded to create "Result3.mxf" in **Figure 41**, all the incoming Material Units are assumed to be recorded for the "Result5.mxf" creation in **Figure 44**.

[60] To be strict, the MpUmids for those MXF files ought to specify the UMID "Live stream" flag to signal their growing during recording as discussed in Section 4.3.4

The FpUmid ("U_b") for the original MXF file is also newly created with a new Mat.# ("bxx") and zero Inst.# values because of multiple sources, none of which is dominant as a main source whose Mat.# is to be inherited. Specifically, if the MXF recorder behaves in a way as discussed in Section 5.4.2, the FpUmid is initially created with the inherited Mat.# ("pxx") value from a UMID ("U_p") attached to the first Material Unit to be recorded as the Edit Unit and non-zero Inst.# value, which is then replaced with a newly created UMID value ("U_b") when the MXF recorder detects a different UMID value ("U_q") attached to the incoming Material Unit after the switching point[61].

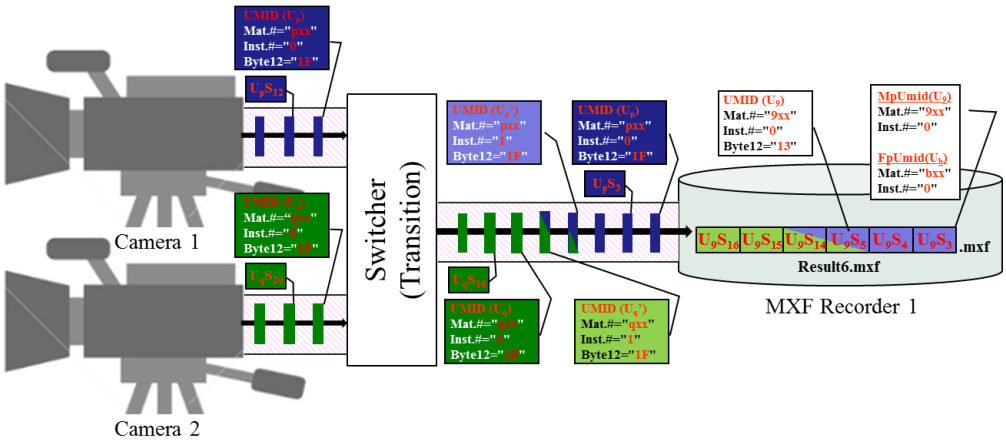


Figure 45: An original MXF file creation from two cameras

Regarding the switching device shown in **Figure 45**, as discussed in Section 5.2, any media processing applied to a Material Unit by the switching device is assumed to result in incrementing the copy number in the Inst.# field of the UMID attached to the Material Unit. Consequently, the UMIDs attached to Material Units within the transition part are replaced with those created by the UMID inheritance with their copy number set to "1", in which the most dominant Material Unit to create a resulting Material Unit is regarded as a main source whose UMID is to be inherited.

Care ought to be taken that the UMID "Live stream" flag for the UMID attached to the resulting Material Unit remains unchanged by the switching device because it does not record the Material Unit at all. Furthermore, all the UMIDs attached to other Material Units than those in the transition part also remain unchanged because of those Material Units going through the switching device as they are without any influence.

5.7.2 From a camera and an MXF player as sources

Figure 46 schematically demonstrates an example in which a stream resulted from the playout of an existing original MXF file ("Source1.mxf") by an MXF player ("MXF Player 2") and a live feed from a camera ("Camera 2") are supplied and switched over at a switching device ("Switcher") to form a single stream, which is then captured by an MXF recorder ("MXF Recorder 1") to create a resulting MXF file ("Result7.mxf").

While the transition effect is also applied to the switching point over two frames in this example, an image overlay such as for a company's logo display at the upper-right corner of the screen is assumed to be applied to all the Material Units that go through the switching device. In addition, in spite of the discussion made in Section 5.3.3, the BodyUmid is intentionally created by the UMID inheritance rather than as a new UMID value equivalent with the MpUmid in this example.

In **Figure 46**, because the resulting MXF file is a newly created one to be managed independently, its MpUmid ("U_{10}") uniquely identifying the MXF file as a whole is newly created with a new Mat.# ("10x") and zero Inst.# values.

Its FpUmid ("U_c") is created in a similar way to those discussed in Section 5.7.1.

---

[61] Although the MXF recorder might detect "U_p' " (attached to the Material Unit just before the switching point) as a different UMID value before "U_q" (attached to that after the switching point) in this example, the inherited UMID value ("U_p' ") is regarded as equivalent with the original UMID value ("U_p") for this comparison because of their common source.

As mentioned above, the BodyUmid is intentionally created by the UMID inheritance with the inherited Mat.# value and the non-zero Inst.# whose copy number is incremented by one due to the lossy compression before recording.

Consequently, the Mat.# value used for a BodyUmid to be assigned to an Edit Unit depends on the source of an incoming Material Unit from which the Edit Unit derives, *i.e.*, the Mat.# of "1xx" and "qxx" are used for the BodyUmids assigned to the Edit Units derived from the Material Units incoming from the MXF player and the camera as sources, respectively. In addition, the Mat.# of a UMID attached to the most dominant Material Unit used to create a resulting Edit Unit is inherited for the creation of BodyUmid to be assigned to the Edit Unit when it is composed of multiple Material Units as sources such as the picture-in-picture (PiP) screen.

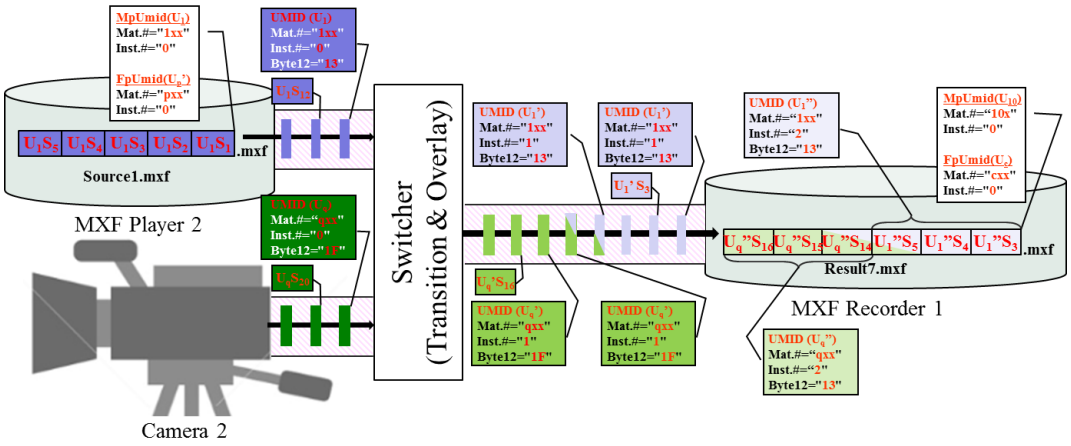


Figure 46: A new MXF file creation from an MXF player and a camera

On the other hand, since the copy number in the Inst.# field of all the UMIDs attached to incoming Material Units has been already set to "1" due to a single media processing executed by the switching device as mentioned below and depicted in **Figure 46**, the copy number for all the BodyUmids is incremented to be "2", while the Source Pack appended to the incoming UMID is always inherited as is.

Note that the UMID "Live stream" flag is always reset to "3_h" for all cases because of its recording as the Edit Units with the Inst.# created by the "Copy number and 16-bit PRS (Pseudo-Random Sequence) generator" method as assumed in Section 5.2, though the UMID "Live stream" flag in the UMIDs attached to the incoming Material Units from the MXF player has been already reset to "3_h" because the original MXF file being played out has already existed persistently in the MXF player.

Regarding the switching device shown in **Figure 46**, because a single media processing for the image overlay is applied to all the Material Units that go through the switching device, the copy numbers in the Inst.# fields of the UMIDs attached to those Material Units are always incremented by one according to the term (f) in Section 5.2[62], while the most dominant Material Unit used to create a resulting composite Material Unit on the wire is regarded as a source whose UMID is to be inherited.

Note that while the image overlay is assumed for the media processing of a Material Unit in this example, any other media processing applicable to a Material Unit such as a transcoding or color grading is acceptable.

It ought to be also noted that because it is a switching device with no recording function, the UMID "Live stream" flag in an incoming UMID is kept as is, *i.e.*, the value of "F_h" for the Material Unit incoming directly from a camera and the value of "3_h" for that resulted from the playout of an original MXF are kept unchanged and transferred as they are.

62 Though it looks that a couple of media processing are applied to the Material Units in the transition part, they all are regarded as a single complex media processing which makes the copy number incremented also just by one.

5.7.3 BodyUmid and the original material

Thanks to the BodyUmid creation by the UMID inheritance, one BodyUmid attached to an Edit Unit is always resolved to the original material from which the Edit Unit initially derives, and the number of media processing that have been applied to the Edit Unit since its origin in the original material is accommodated as the copy number in the Inst.# field of the BodyUmid, as indicated by the BodyUmid "U_1" ", in **Figure 46**.

There is another BodyUmid, on the other hand, whose resolution cannot lead to the original material because of its absence at the time of the resolution as shown by the BodyUmid "U_p" " also in the figure.

Unfortunately, whether the original material exists or not cannot be determined until when a given BodyUmid is made a trial to resolve. Furthermore, whether an original material actually exists but somewhere unknown or it does not exist at all due to its temporary existence on the wire cannot be distinguished when the BodyUmid is unresolved[63].

Therefore, as already discussed in Section 5.3.3, the BodyUmid to be assigned to an Edit Unit is primarily recommended to be newly created as equivalent with the MpUmid uniquely identifying an MXF file that contains the Edit Unit for the new MXF file creation. In particular, such a BodyUmid creation is most preferable for the MXF file that contains an Edit Unit being created in a persistent form for the first time[64] such as that assigned with the BodyUmid "U_p" " shown in **Figure 46**, in order to provide an original persistent "Edit Unit" to be practically referenced back from any descendant Edit Unit whose direct descendant was initially derived from the original in any way.

On the other hand, it is interesting to note that if a newly created MXF file is previously known to contain such an Edit Unit as being created in a persistent form for the first time because of, *e.g.*, the system configuration for the file ingest, there is another way worthwhile to be considered.

If this is the case, such an Edit Unit contained in an MXF file being newly created is obvious to be regarded as an original one regardless of the kinds of BodyUmid to be assigned to it. Hence, even if the BodyUmid is unresolved, the Edit Unit is understood that it is not because of unknown location of its origin but because of the BodyUmid inherited from the UMID attached to an incoming Material Unit temporarily existing on the wire[65].

Hence, the problem to be solved is reduced to how to prevent the Mat.# of such a BodyUmid as having its origin only temporarily existing on the wire from being propagated over the media production workflow chain so that any descendant Edit Unit derived from the original material and contained in a certain derived material created somewhere along with the workflow chain can be referenced back to the original material containing its origin.

In fact, Section 4.5.3.5 has already provided the solution for it, *i.e.*, if the basic part of BodyUmid is replaced with an MpUmid when it is attached to a Material Unit to be transferred during the playout of a new MXF file uniquely identified by the MpUmid as shown in **Figure 34** (b), the MXF recorder will receive and record the Material Unit as the Edit Unit whose BodyUmid is to be resolved to the MpUmid of the new MXF file. In other words, if an MXF player that can receive the live feed and create a new MXF file in it has a function to replace the basic part of BodyUmid with an MpUmid at the playout of an MXF file uniquely identified by the MpUmid, the MXF file played out by the MXF player is always regarded as an original MXF file which is to be referenced back from any descendent material derived from the original MXF file in any way in any step of the media production workflow chain.

---

[63] Since the original material is unavailable in both cases anyway, there is no substantial difference between them in the practical situation, though.

[64] Because the item (i) in Section 5.2 assumes that a single BodyUmid treatment must be applied to all the Edit Units contained in an MXF file, if such an Edit Unit as to be recorded for the first time is found to be contained in an MXF file to be newly created, all the Edit Units in the MXF file are assigned with the BodyUmid whose basic part is equivalent with the MpUmid uniquely identifying the MXF file as a whole.

[65] Following the discussion made in Section 4.4.3.5, the BodyUmid, when unresolved to the original material, is regarded as the material "source" identifier if the material "source" appropriately manages a UMID to be attached to the Material Unit it creates.

References

1 SMPTE ST 330:2011 – Unique Material Identifier (UMID)

2 SMPTE RP 205:2014 – Application of Unique Material Identifiers in Production and Broadcast Environments

3 EBU Technical Review Special Supplement 1998, “EBU / SMPTE Task Force for Harmonized Standards for the Exchange of Programme Material as Bitstreams, Final Report: Analyses and Results”: http://tech.ebu.ch/docs/techreview/ebu-smpte-tf-bitstreams.pdf

4 SMPTE ST 377-1:2011 – Material Exchange Format (MXF) – File Format Specification

5 Y. Shibata and Jim Wilkinson, “UMID Applications in Practice”, SMPTE Mot. Imag. J.; March 2012; 121:(2) 58-67

6 “STUDY REPORT ON UMID APPLICATIONS PART 2-1 - Additional Technologies Needed to be Standardized (1)”, SMPTE TC30MR Study Report, available at: https://kws.smpte.org/kws/groups/30mr/download/27420

7 SMPTE EG 377-3:2013 – Material Exchange Format (MXF) – Engineering Guideline (Informative)

8 SMPTE ST 378:2004 for Television – Material Exchange Format (MXF) – Operational Pattern 1a (Single Item, Single Package)

9 SMPTE ST 392:2004 for Television – Material Exchange Format (MXF) – Operational Pattern 2a (Play-List Items, Single Package)

10 SMPTE ST 391:2004 for Television – Material Exchange Format (MXF) – Operational Pattern 1b (Single Item, Ganged Packages)

11 AMWA AS-02 MXF Versioning
available at: http://amwa.tv/downloads/specifications/AMWA-AS-02-10-2011-11-18_MXF_Versioning.pdf

12 SMPTE ST 390:2011 – Material Exchange Format (MXF) – Specialized Operational Pattern “OP-Atom” (Simplified Representation of a Single Item)

13 SMPTE ST 408:2006 for Television – Material Exchange Format (MXF) – Operational Patterns 1c, 2c and 3c

14 SMPTE ST 379-1:2009 – Material Exchange Format (MXF) — MXF Generic Container

15 SMPTE ST 379-2:2010 for Television – Material Exchange Format (MXF) – MXF Constrained Generic Container

16 SMPTE ST 326:2000 for Television – SDTI Content Package Format (SDTI-CP)

17 SMPTE ST 385:2012 for Television – Material Exchange Format (MXF) – Mapping SDTI-CP Essence and Metadata into the MXF Generic Container

18 SMPTE ST 331:2011 – Element and Metadata Definitions for the SDTI-CP

19 SMPTE ST 291-1:2011 – Ancillary Data Packet and Space Formatting

 Approved 11 December 2015